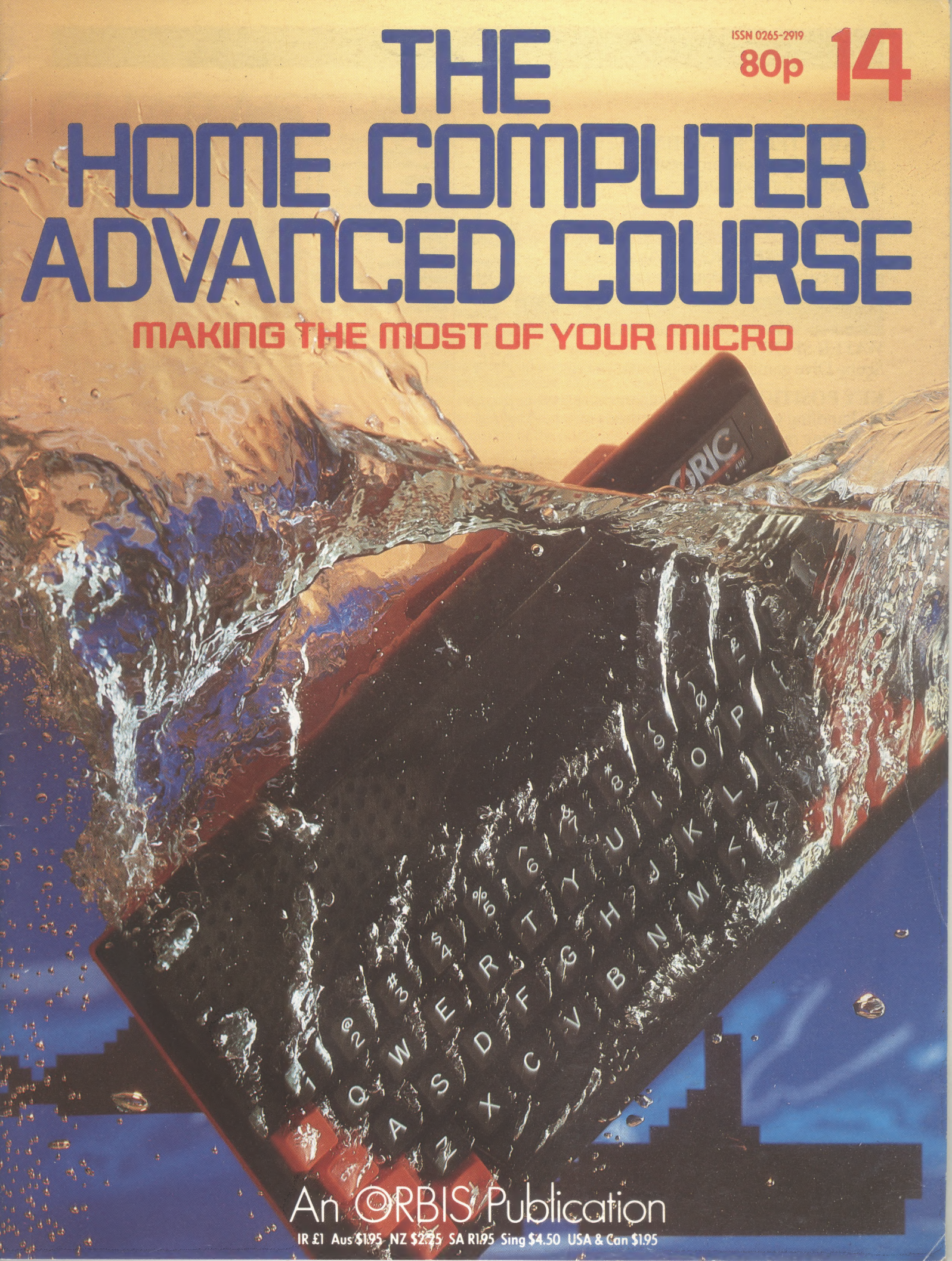
ISSN 0265-2919

80p 14

THE HOME COMPUTER ADVANCED COURSE

MAKING THE MOST OF YOUR MICRO



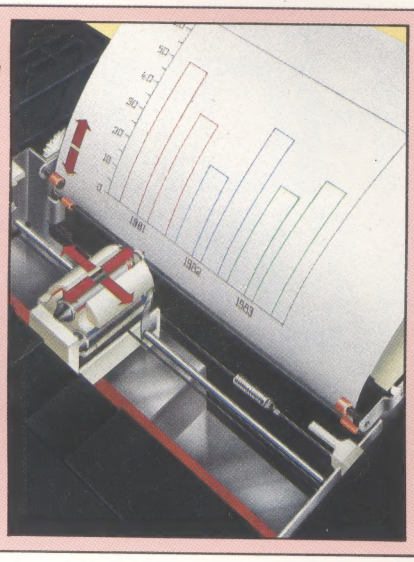
An ORBIS Publication

IR £1 Aus $1.95 NZ $2.25 SA R1.95 Sing $4.50 USA & Can $1.95

CONTENTS

Next Week

- **We review a selection of low cost colour printer/plotters and explain their working principles.**
- **Our software classic is Quicksilva's intriguing Boogaboo cartoon game for the Commodore 64 and the Spectrum.**
- **We complete our graphics course for the Commodore 64. In future weeks we will set more projects for a selection of the popular micros.**
- **We look at the practicality of running a database on a home micro and warn against some of the common pitfalls.**



QUIZ

1) What advantage do the Z80 registers have over the 6502?

2) What is the effect of simultaneously pressing the 'Z' and Function keys on the Oric Atmos?

3) How many points do you score for catching an electric eel in Scuba Dive?

4) In computer programs, user RAM is generally at a premium. Why, therefore, do we have to reduce the top of the Basic Text Area in the Subhunter program?

Answers To Last Week's Quiz

A1) Line 225 is forbidden to the programmmer in BASICODE and CLS is replaced by the command GOSUB 100.

A2) Records are the same length to allow the operating system to position the read/write head in the correct place.

A3) The process is known as 'bit-mapping'.

A4) The fact that the machine runs on a Z80A microprocessor allows it to use software that runs under CP/M.

QUIZ

COVER PHOTOGRAPHY BY CHRIS STEVENS

Editor Jim Lennox; **Art Director** David Whelan; **Technical Editor** Brian Morris; **Production Editor** Catherine Cardwell; **Picture Editor** Claudia Zeff; **Sub Editor** Robert Pickering; **Designer** Julian Dorr; **Art Assistant** Liz Dixon; **Editorial Assistant** Stephen Malone; **Contributors** Steven Colwill, Max Phillips, David Janda, Richard Pawson; **Group Art Director** Perry Neville; **Managing Director** Stephen England; **Published by Orbis Publishing Ltd: Editorial Director** Brian Innes; **Project Development** Peter Brookesmith; **Executive Editor** Chris Cooper; **Production Co-ordinator** Ian Paton; **Circulation Director** David Breed; **Marketing Directnr** Michael Jovce: **Designed and produced by Bunch Partworks Ltd; Editorial Office** 85 Charlotte Street, London W1P 1LB;

HOME COMPUTER ADVANCED COURSE – Price UK 80p IR £1.00 AUS $1.95 NZ $2.25 SA R1.95 SINGAPORE $4.50 USA and CANADA $1.95

How to obtain your copies of HOME COMPUTER ADVANCED COURSE – Copies are obtainable by placing a regular order at your newsagent, or by taking out a subscription. Subscription rates: for six months (26 issues) £23.80; for one year (52 issues) £47.60. Send your order and remittance to Punch Subscription Services, Watling Street, Bletchley, Milton Keynes, Bucks MK2 2BW, being sure to state the number of the first issue required.

Back Numbers UK and Eire – Back numbers are obtainable from your newsagent or from HOME COMPUTER ADVANCED COURSE. Back numbers, Orbis Publishing Limited, 20/22 Bedfordbury, LONDON WC2N 4BT at cover price. AUSTRALIA: Back numbers are obtainable from HOME COMPUTER ADVANCED COURSE. Back numbers, Gordon & Gotch (Aus) Ltd, 114 William Street, PO Box 767G Melbourne, Vic 3001. SOUTH AFRICA, NEW ZEALAND, EUROPE & MALTA: Back numbers are available at cover price from your newsagent. In case of difficulty write to the address in your country given for binders. South African readers should add sales tax.

How to obtain binders for HOME COMPUTER ADVANCED COURSE – UK and Eire: Please send £3.95 per binder if you do not wish to take advantage of our special offer detailed in Issues 5, 6 and 7. EUROPE: Write with remittance of £5.00 per binder (incl. p&p) payable to Orbis Publishing Limited, 20/22 Bedfordbury, LONDON WC2N 4BT. MALTA: Binders are obtainable through your local newsagent price £3.95. In case of difficulty write to HOME COMPUTER ADVANCED COURSE BINDERS, Miller (Malta) Ltd, M.A. Vassalli Street, Valletta, Malta. AUSTRALIA: For details of how to obtain your binders see inserts in early issues or write to HOME COMPUTER ADVANCED COURSE BINDERS, First Post Pty Ltd, 23 Chandos Street, St. Leonards, NSW 2065. The binders supplied are those illustrated in the magazine. NEW ZEALAND: Binders are available through your local newsagent or from HOME COMPUTER ADVANCED COURSE BINDERS, Gordon & Gotch (NZ) Ltd, PO Box 1595, Wellington. SOUTH AFRICA: Binders are available through any branch of Central Newsagency. In case of difficulty write to HOME COMPUTER ADVANCED COURSE BINDERS, Intermag, PO Box 57394, Springfield 2137.

Note – Binders and back numbers are obtainable subject to availability of stocks. Whilst every attempt is made to keep the price of the issues and binders constant, the publishers reserve the right to increase the stated prices at any time when circumstances dictate. Binders depicted in this publication are those produced for the UK market only and may not necessarily be identical to binders produced for sale outside the UK. Binders and issues may be subject to import duty and/or local taxes, which are not included in the above prices unless stated.

CHARACTER CRUNCHING

By far the most popular serious use for microcomputers is word processing. Most people need to produce documents such as letters, reports or essays from time to time, and a word processor makes all of these tasks much easier. We take a detailed overview of the subject, and discuss some of the more popular packages.

Word processors are really nothing more than computerised versions of the typewriter. Text is keyed in through the computer keyboard and appears on the screen. Changes can be made easily without the need for retyping the whole document and once the wording is correct, the text is printed out by a computer printer.

Apart from a general fear of computers, the only thing likely to put people off beginning to use a word processor is the problem of not feeling at home using a keyboard. Yet it is easier to get started using a keyboard on a word processor than on an ordinary typewriter. The inevitable mistakes of the fumbling two-fingered typist having a first go on a typewriter can create an awful mess. These can be put right in seconds on a word processor to produce copy that is perfect the first time it is printed — and that is a real confidence builder.

Just about any home micro can be used for word processing, but some are not as well suited as others. In some cases this is because good word processing software is not available for the particular machine; in other cases it is the micro and its peripherals that are not suitable to the task.

The cost of even a simple word processing system can be quite high, as the necessary extras can easily cost twice as much as the computer itself. Usually the most expensive single item is a printer. Without access to a good printer there is little point in having a word processor package. For the foreseeable future just about all word processed text is going to end up printed on paper — the age of electronic mail, where all text is sent directly from one micro to another, is still a long way off.

Even the simplest printers are fairly expensive, and yet the quality of the printing they produce is relatively poor. In many cases word processing demands high quality printing. After all, there's little point in spending your time with a word processor getting the wording of a job application letter just right if the result is then printed out on a dot matrix printer. Daisy wheel printers offer better quality printing but are slow and expensive, although prices are coming down quite quickly. Some electronic typewriters can have interfaces fitted to them so that they can be used as computer printers. This may help save money for people switching from typing to word processing.

One solution to the dilemma of gaining access to a quality printer is for several friends or a computer club to share the cost of the machine between them. Users would still have the problem of interfacing their micros to use the printer. With some computers this is easy because they have standard interfaces, so each user will only need to buy a suitable cable to link the two together. Other micros, such as the Commodore and Atari machines, have interfaces that limit them to their

BBC Micro
The combination of BBC Micro and Torch disk pack allows business software to be used, including the excellent Wordstar. The cost of the system is, however, higher than many business micros. Admittedly a cheaper printer could be used, and the Perfect Writer software that is included in the price of the Torch disk pack could be used instead of buying Wordstar. This would bring the price down to about £1,500

BBC Micro	**£399**
Torch disk pack	**£804**
Wordstar	**£340**
Silver Reed EXP 770 Printer	**£1,024**
Printer Cable	**£10**
TOTAL	**£2,577**

IAN McKINNELL

own brand of printers. A few micros, including the Sinclair Spectrum, have no printer interface as such, and so an interface add-on has to be bought.

With a suitable printer available, your next task is to choose the appropriate processing software. A wide range of programs is produced for the more popular makes of computer, while only one or two are available for the less popular machines. The quality varies considerably between different programs. Some are crude and allow only simple editing, such as inserting and deleting text. Others allow whole passages to be moved around within the article, present the text on the screen just as it will appear on paper, or justify the text (adjust the word spacing so that the line length is uniform, like the text you are now reading).

Some word processors can search for a particular word or phrase, so a spelling mistake that has been repeated throughout an article can easily be put right. Programs to check the spelling of every word in a piece of text are sold for certain word processors; and it is possible for other programs, such as a mailing list or a database, to work in conjunction with the word processor. More sophisticated word processing programs are designed to make use of the features of certain printers. Dot matrix printers can often produce several different types of printface (such as italicised, bold or small letters) and so some word processors allow the different types to be mixed in one article. A few word processors can be expanded to use the ability of certain dot matrix printers to produce graphics. This allows many new typefaces to be used, including letters larger than normal and various ornate styles such as copperplate handwriting. Used in moderation these can liven up the printed text considerably.

Daisy wheel printers can use 'proportional spacing' to give more space to wide letters such as 'w' and less to narrow ones like 'i', rather than allowing them all the same space as an ordinary typewriter would do. Some word processors can use this facility, which makes the text much more readable, and yet still manage to 'justify' both margins. This is ideal for producing community newspapers or club magazines because it gives a professional look without incurring the expense of proper typesetting. The interchangeable print wheels allow various typefaces to be chosen to suit the article. Alternatively, some typesetters will accept word processed copy on floppy disk or even tape. This gives top quality results without the cost of having someone key all the text into a typesetting machine.

Word processing software is sold in various formats including tape, disk, cartridge and ROM chip. More important, however, is the way word processed text is stored — usually on tape or floppy disk. Although cheap, tape is awkward, slow and limits the length of articles to the size that can be held in memory. Disks are better because they are fast, reliable and allow long articles to be written. New ways of storing data are starting to appear. The Sinclair Microdrive, for example, is cheap yet can store large amounts of data and find a specified part of the text in seconds. However, few word processors for the Spectrum are able to work with the Microdrives yet. Another interesting system is the tape drive used by the Coleco Adam, a home micro obviously designed with word processing in mind because it includes a daisy wheel printer. It uses modified cassette tapes to store its data and these can find any item within a few seconds.

Saving word processed copy on tape or disk allows long articles to be written over several days, standard letters to be used many times and copies of all work to be kept. It's also a good idea to make copies of long items at various stages while they are being written. If this isn't done there is a danger that some accident such as a power cut will destroy all the work.

Some programs have odd commands and awkward key combinations to memorise, while

Sinclair Spectrum
This is the cheapest efficient word processing system, but it is still quite expensive. Several limitations are imposed by the Spectrum, including a poor keyboard, the lack of a monitor interface and the provision of Microdrives instead of disk drives. However, it would be possible to add a better quality keyboard. Tasword Two is one of the few Spectrum word processors that will work with the Microdrives

Sinclair Spectrum (48K)	**£130**
Interface 1	**£50**
Two Microdrives	**£100**
Tasword Two software	**£14**
Shinwa CP80 Printer	**£230**
RS232 adaptor for CP80	**£60**
Printer cable	**£15**
TOTAL	**£599**

IAN McKINNELL

others are straightforward. Some micros, such as the Sinclair Spectrum, have poor quality keyboards that make life a misery. At least there are add-on keyboards for the Spectrum that bring it up to an acceptable standard. Keyboards with extra function keys are good because they cut down on the number of command codes that have to be memorised. The screen display can also cause problems. Several micros show only a small amount of text on screen at once, which makes writing much harder. The Commodore Vic-20, for example, can only show text 22 characters wide, whereas business machines usually have an 80-character screen width. The ideal micro for heavy use is one that gives at least a 25-by-80 character text display and can use a proper monitor to give a clear sharp image.

The design of the letters shown on the computer screen varies considerably. Some machines make up each letter from more dots than others, which makes the display easier to read and work with. A few home micros use letters only six dots wide while most use an eight dot wide grid. A few business machines have remarkable 16-by-16 dot characters, which are of superb quality. Some people find it hard to work with a screen and have to print the text out on paper before they can read through it.

Modest home systems are suitable for writing letters and other short items, but more equipment is needed to make writing books or long reports practical. If you need to do a lot of word processing, you will need a system with a monitor, two disk drives, a good printer, a typewriter-style keyboard and a good piece of word processing software. Expanding a home micro to this level is expensive — often more expensive, in fact, than buying a true business micro.

Business machines offer other advantages. Because they were designed for business needs, they have good keyboards, screens, disk drives and printer interfaces. Even so, their most important advantage is the high quality software available for them. There is such a range of good software available because most business computers use one of a few standard operating systems, which means each word processor program is available for many different machines.

By far the best known business word processing package is Wordstar, which is available for the CP/M, CP/M-86 and MS-DOS operating systems. The program has some sophisticated features but is expensive, costing over 20 times as much as the average home micro program. The cost of a business word processor is high but it is a false economy for a small business, or even a serious writer, to use a home micro for large amounts of word processing. The time wasted using a limited system will soon outweigh any money saved.

The cost of serious business systems is high, but it is coming down all the time. Some home micros can be expanded to use standard operating systems such as CP/M, so people who have already invested a lot in a home micro system are beginning to be able to use serious software without having to pay too much extra. Another trend is helping to bring down the cost of serious word processing. Several companies are including word processing programs such as Wordstar free of charge with their computers. Some companies, however, give away programs that are not up to Wordstar's standards.

The newest fad to hit word processing is word processing on the move. Several battery-powered computers are sold with built-in word processors to allow busy executives to write memos and letters anywhere at any time. These machines are beyond the pocket of most home micro users and do not have many other uses. But hand-held machines do seem to suggest that word processing is becoming more and more common. Maybe not only the typewriter is doomed to extinction, but pen and paper as well.

Commodore 64
The Commodore 64 offers the most reasonably priced word processor with a disk drive. Having only one disk drive is limiting and the Commodore 64 can produce only a 40-character wide display. A cheaper Commodore printer is available at about £200 but its print quality is very poor

Commodore 64	**£200**
Commodore disk drive	**£200**
Easy Script software	**£75**
Commodore 1526 printer	**£345**
TOTAL	**£820**

IAN McKINNELL

ASSEMBLING THE CAST

Sprite creation is one of the most exciting features of the Commodore 64's graphics capabilities — both in terms of the enjoyment gained in designing them and because they allow fast-moving games to be written in BASIC. In this part of our project to create a Subhunter game, we take you through the full procedure of sprite design.

A sprite is a large movable graphics shape. It is designed in much the same way as the eight-by-eight user-defined characters discussed earlier in the course (see page 233) but is constructed on a much larger grid. Once a sprite is defined, attributes such as colour and screen position are controlled by a set of special registers in the Commodore 64's video control (or VIC) chip.

A sprite is made up of 21 rows of 24 pixels. Each row consists of three eight-pixel segments and is represented by three bytes of memory, so 63 bytes in all are required to store the data for one sprite. As with user-defined characters, each pixel on the sprite grid that will be illuminated in the final sprite shape is represented by binary one (and the non-illuminated pixels by binary zero). Thus, for each row of the sprite we can calculate the decimal equivalents of each group of eight binary digits. The diagrams we give here show the four sprites that will be used in the Subhunter game. The numbers down the side of each drawing are the decimal equivalents that will form the data statements for each sprite (as given in lines 6000 to 6370 of the program).

LOWERING THE TOP OF MEMORY

Once a sprite has been defined and converted into a series of DATA statements, the data must be READ and POKEd into memory. Sprite data can be positioned in several memory locations. For example, using the locations starting at 12288 will place it in the BASIC program's area, which runs from 2048 to 40960. As a BASIC program is typed into the Commodore 64, it fills memory space from location 2048 onwards. A program would need to be 10 Kbytes long before it would reach location 12288 and thus overwrite the sprite data. However, when a program is running, any variables used are stored in the area above that used to store the program — string variables, in particular, build down from the top of the BASIC program area. As the Subhunter game uses the timer variable, TI$, regular updating of its value and its subsequent storage will eventually overwrite the area in which we wish to store the sprite data.

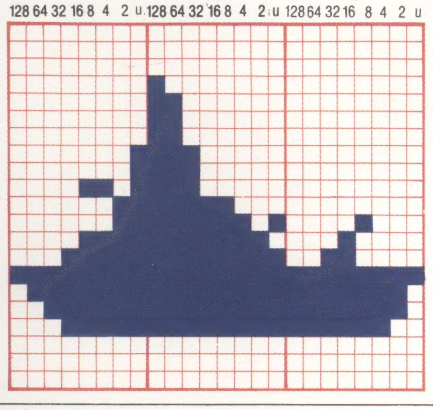
A solution to this problem is to lower the top of the BASIC program area to below the area in which the sprite data is kept. The pointer to the address of the top of memory is held at locations 55 (lo-byte) and 56 (hi-byte). Normally these two locations contain the values 0 and 160 respectively, which represent the address 40960. In lo-hi form, the location 12288 is given as 0 and 48. We can lower the top of memory to this location by simply POKEing these values into locations 55 and 56 at the start of the program (see line 90).

SPRITE POINTERS

As sprite data can be positioned in various parts of memory, a pointer is needed to indicate where that data begins. There are eight sprite pointers, held in locations 2040 (for sprite 0) to 2047 (for sprite 7). The value held in each sprite pointer is related to the area that holds the sprite data by this formula: start of 63 bytes of data = (sprite pointer)×64. The data for the ship in our program starts at 12288 and the ship is to be designated sprite 0, so the pointer in location 2040 is 192

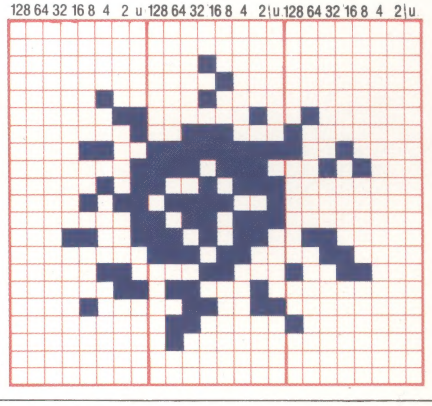
Moving Target
A sprite is made up from 21 rows of three bytes; these bytes are actually binary bit patterns, but are stored in the BASIC program data lines as their decimal number equivalents. These values can be seen beside the sprite diagrams and in the program listing. The program POKEs the values into a dedicated area of RAM, where the video controller chip accesses them as sprite data, displaying and moving them on the screen with a minimum of programming effort

SHIP – SPRITE 0



1	2	3
0	0	0
0	0	0
0	0	0
0	128	0
0	192	0
0	192	0
0	192	0
1	224	0
1	224	0
13	224	0
3	248	128
3	253	8
15	254	16
31	255	48
255	255	255
127	255	254
63	255	254
31	255	252
0	0	0
0	0	0
0	0	0

EXPLOSION – SPRITE 1



1	2	3
0	0	0
0	0	0
0	16	0
0	8	0
4	16	0
3	2	64
1	56	128
12	255	144
1	238	40
5	151	0
11	121	0
1	183	0
25	214	96
0	236	48
6	24	152
3	98	0
8	51	0
0	96	128
0	64	0
0	0	0
0	0	0

(12288/64). The next block of data is for the explosion, sprite 1. If we set the pointer in location 2041 to 193, then the data must start at 12352. These are the values we are using:

Sprite Number	0	1	2	3
Sprite Pointer	192	193	194	195
63 Bytes Of Sprite Data	12288 to 12350	12352 to 12414	12416 to 12478	12480 to 12542

Notice that one byte remains unused at the end of each block of sprite data. The parts of the program listing that read the sprite data of memory and specify the sprite pointers are contained in lines 2000 to 2210.

MANIPULATING SPRITES

The VIdeo Control (VIC) chip has several special registers that are used to control sprites. The first location of the VIC chip is 53248, and it is simpler for our program to describe the locations of all the other registers as relative to this. If we let V=53248, the next location of the VIC chip, 53249, can be termed V+1 and so on. V should be defined, with other variables, at an early stage (see line 100).

The colour of each sprite is set by POKEing a colour code number (in the range 0 to 15) into a special register. Each of the eight sprites has its own colour register; these run from V+39 to V+46. For example, to colour the ship black we simply POKE the colour code 0 into location V+39. The other sprites can be coloured in the same way (see lines 2220 to 2250).

Positioning sprites on the screen will be discussed in greater detail in the next instalment. For now it is sufficient to know that the x co-ordinate of sprite 0 is held in location V, the y co-ordinate for sprite 0 is held in location V+1 ; the x and y co-ordinates for sprite 1 are held in V+2 and V+3 respectively, and so on up to location V+15 (see lines 2260 to 2280).

Sprites can be expanded horizontally, vertically, or in both directions, by a factor of two. The ship and sub sprites may seem rather squashed horizontally, but we will now expand them to twice their original length. In fact all four sprites will be expanded horizontally. The VIC chip register controlling horizontal expansion is V+29, which is easier to use than the other registers we have discussed. Instead of using eight different registers to control the attributes of each of the eight sprites, all that is required is to switch the function on or off. Therefore only one bit within the register is required to control the horizontal expansion for each sprite. If a sprite is to be expanded horizontally, the corresponding bit in the V+29 register must be set to 1. The following table shows the POKE required to expand all the four sprites we have defined:

Sprite Number	7	6	5	4	3	2	1	0
Contents Of V+29	0	0	0	0	1	1	1	1

= 15 (decimal)

Expansion in the vertical direction is controlled by V+23. The explosion, sprite 1, is expanded vertically and horizontally, thus doubling its size (see lines 2290 to 2310).

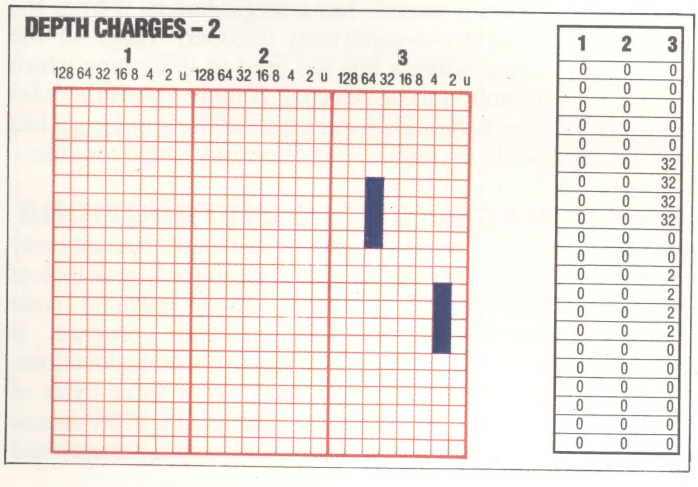
Our final task is to turn the required sprites on. A single bit in the VIC chip register, V+21, is used to switch each sprite on or off. In the Subhunter game only the ship and the sub are initially turned on (lines 2310 to 2360).

Once you have typed all of the listing in, you should test that the sprite data has been read correctly. To do this, run the program and break into it using RUN or STOP when the timer appears at the top of the screen. Entering the following statements, without line numbers, will position and display all four sprites created by the routine.

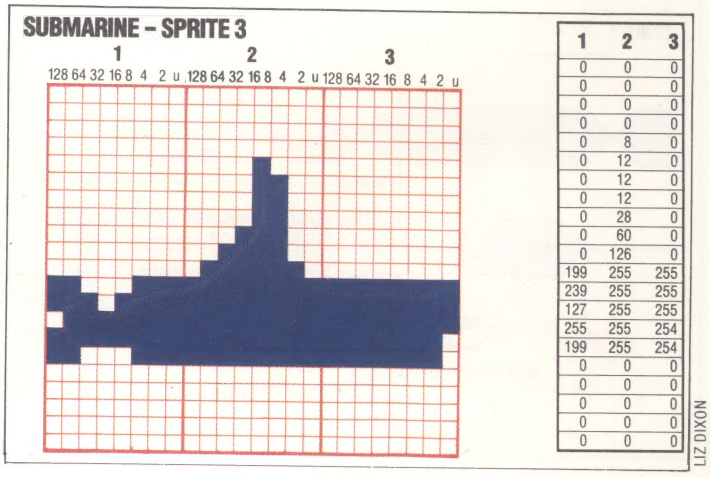
```
POKEV,160                (Ship's co-ordinate)
POKEV+2,240:             (Explosion's x and y
  POKEV+3,100             co-ordinates)
POKEV+4,160:             (Depth charge's
  POKEV+5,100             x and y co-ordinates)
POKEV+6,100:             (Submarine's x and y
  POKEV+7,100             co-ordinates)
POKEV+21,15              (Turns on sprites 0 - 3
```

If the program stops with an 'OUT OF DATA ERROR' message, check how many numbers there are in the DATA statements. There should be 63 for each sprite. If the program crashes and the keyboard fails to respond, make sure that V has been declared in line 100. It is always a good idea to SAVE your program before running it.

```
1 REM***C64 GRAPHICS*********
90 POKE 55,0:POKE 56,48:CLR
:REM LOWER MEMTOP
100 V=53248:FL=0:SC=0
110 GOSUB 1000
:REM SCREEN SETUP (see p215)
120 GOSUB 2000
:REM SPRITE CREATION
2000 REM***SPRITE CREATION***
2020 REM** READ SHIP DATA  **
2030 FOR I=12288 TO 12350
2040 READ A:POKE I,A:NEXT I
2060 REM** READ EXP  DATA  **
2070 FOR I=12352 TO 12414
2080 READ A:POKE I,A:NEXT I
2100 REM** READ CHRG DATA  **
2110 FOR I=12416 TO 12478
2120 READ A:POKE I,A:NEXT I
2140 REM** READ SUB DATA   **
2150 FOR I=12480 TO 12542
2160 READ A:POKE I,A:NEXT I
2180 REM** SET POINTERS    **
2190 POKE 2040,192:POKE 2041,
193:POKE 2042,194:POKE
2043,195
2220 REM** SET COLOURS      **
2230 POKE V+39,0:POKE V+40,1:
POKE V+41,0:POKE V+42,0
2260 REM**INIT SHIP COORDS **
2270 POKE V+1,80:X0=160
2290 REM** EXPAND SPRITES  **
2300 POKE V+29,15:POKE V+23,2
2320 REM** TURN ON SPRITES **
2330 POKE V+21,9
2340 RETURN
2350:
6000 REM** SHIP DATA       **
6010 DATA 0,0,0,0,0,0,0,0,0
6020 DATA 0,128,0,0,192,0,0,
192,0
6030 DATA 0,192,0,1,224,0,1,
224,0
6040 DATA 13,224,0,3,248,128,
3,253,8
6050 DATA 15,254,16,31,255,48,
255,255,255
6060 DATA 127,225,254,63,255,
254,31,255,252
6070 DATA 0,0,0,0,0,0,0,0,0
6100 REM** EXPLODE DATA    **
6110 DATA 0,0,0,0,0,0,0,16,0,
0,8,0,4,16
6120 DATA 0,3,2,64,1,56,128,
12,255,144
6130 DATA 1,238,40,5,151,0,11,
121,0,1
6140 DATA 183,0,25,214,96,0,
236,48,6,24
6150 DATA 152,3,98,0,8,51,0,
0,96,128,0
6160 DATA 64,0,0,0,0,0,0,0,0
6200 REM** DEPTH CHRG DATA **
6210 DATA 0,0,0,0,0,0,0,0,0,
0,0,0,0,0
6220 DATA 0,0,0,32,0,0,32,0,
0,32,0,0,32,0
6230 DATA 0,0,0,0,0,0,0,0,0
6240 DATA 2,0,0,2,0,0,2,0,0,
2,0,0
6250 DATA 0,0,0,0,0,0,0,0,0
6260 DATA 0,0,0,0,0,0,0,0,0
6300 REM** SUBMARINE DATA  **
6310 DATA 0,0,0,0,0,0,0,0,0,
0,0,0
6320 DATA 0,8,0,0,12,0,0,12,0
6330 DATA 0,12,0,0,28,0,0,60,0
6340 DATA 0,126,0,199,255,255
6350 DATA 239,255,255,127,
255,255
6360 DATA 255,255,254,199,
255,254
6370 DATA 0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0
```



1	2	3
0	0	0
0	0	0
0	0	0
0	0	0
0	0	0
0	0	32
0	0	32
0	0	32
0	0	32
0	0	0
0	0	0
0	0	2
0	0	2
0	0	2
0	0	2
0	0	0
0	0	0
0	0	0
0	0	0
0	0	0
0	0	0



1	2	3
0	0	0
0	0	0
0	0	0
0	0	0
0	8	0
0	12	0
0	12	0
0	12	0
0	28	0
0	60	0
0	126	0
199	255	255
239	255	255
127	255	255
255	255	254
199	255	254
0	0	0
0	0	0
0	0	0
0	0	0
0	0	0

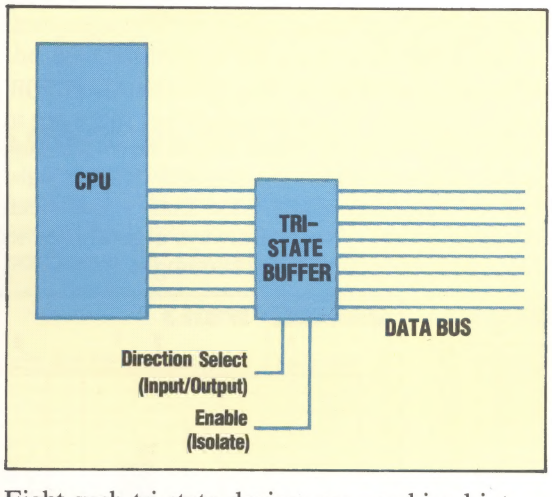
LIZ DIXON

MEMORY JOGGING

The central processing unit or CPU is the nerve centre of your home computer. In this instalment of the Computer Science course, we take a look at the logic of data transfer between the CPU and memory. In this discussion, we introduce the address and data buses, tri-state devices and the memory address register.

Each memory location in a home computer is normally made up of eight bits. Data can be transferred, eight bits at a time, along a series of eight parallel lines to the CPU where the data can then be used according to a program instruction. Data can also be sent in the opposite direction in order to be stored in a memory location. The machine code instruction LDA $1234 causes the number in location $1234 to be sent along the data bus to the CPU. STA $1234 causes a number to be sent from the CPU, again down the data bus, and stored in location $1234.

The data bus must therefore allow transfers in both directions. At certain times it is also important to isolate the CPU from the data bus. Thus each line of the data bus can be in one of three states (INPUT, OUTPUT or ISOLATE). In order to achieve the necessary switching between these states, each data bus line has a small electronic circuit known as a *tri-state device.*



Eight such tri-state devices are combined into a single integrated circuit. The diagram above shows how this integrated circuit links the data bus with the CPU. The diagram also shows the 'enable' and 'direction select' lines, which put the eight tri-states into the required operative state. Such circuits can also be used for connecting other devices, such as input/output peripherals, to the data bus.

Whenever we wish to call up the contents of a particular location we refer to the location required by its address. Each location in ROM and RAM has its own unique number that refers to it. Let us now look at how, at the hardware level, any location in memory can be accessed so that a data transfer can be accomplished.

Most microcomputers have a second highway between the CPU and memory called the *address bus.* Normally, the address bus has 16 lines rather than eight. This means that up to 65,536 separate addresses can be specified ($2^{16} = 65,536$). That is, up to 64 Kbytes of memory can be accessed using a 16-bit address bus. The total memory area can be thought of as being broken up into modules, each containing 256 locations. The lower eight bits of the address can then be used to find the particular location within a given module. The module itself may be selected by using some or all of the remaining eight address bits.

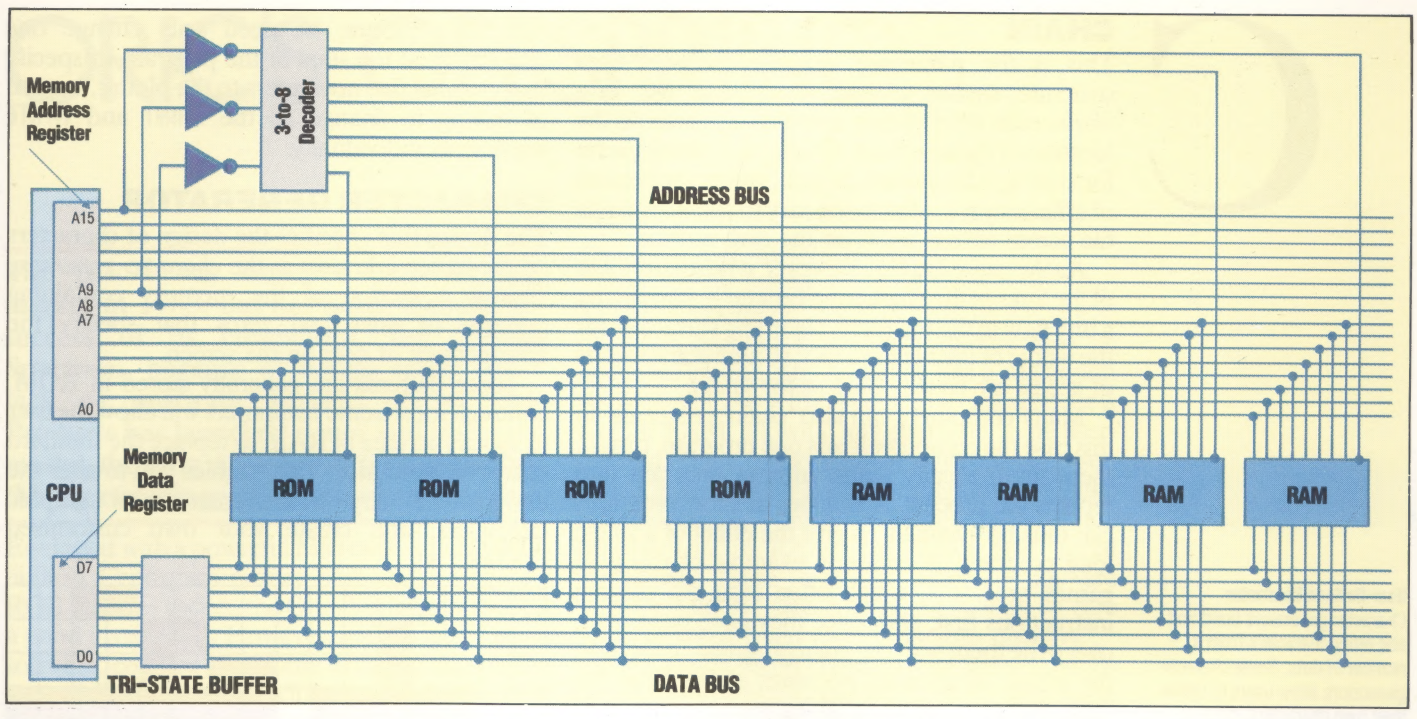
If we consider the simple example of a microcomputer with a total memory size of two Kbytes, we can see how the selection of any particular location takes place. As each module of memory contains 256 locations, our two-Kbyte computer will require eight modules. For our simple computer we will assume that the memory is equally divided between ROM and RAM.

The address of the required location is held in a special 16-bit register in the CPU, called the memory address register or MAR. As the lower eight bits of the address select a particular location within any module, the lower eight lines of the address bus can be connected to each of the memory modules. In order to select a particular module we now require only a further three bits ($2^3 = 8$). This three-bit code must be decoded into eight output lines, one for each module.

The diagram shows how memory modules link via these data and address buses to the CPU. Each memory module has a single line to it from the three(bit)-to-eight(line) decoder. Three of the higher address bits are used to determine which module is to be selected. If more RAM modules were to be added, then more of the upper eight bits would be required to select any single module.

MACHINE CODE INSTRUCTIONS

Having seen how a particular location in memory may be selected and data transferred, let us look at how the CPU carries out a machine code instruction. Any machine code program is normally stored in consecutive storage locations. One instruction may take two or three bytes of storage. An instruction such as ADD $13FF means 'add the contents of the location with hexadecimal

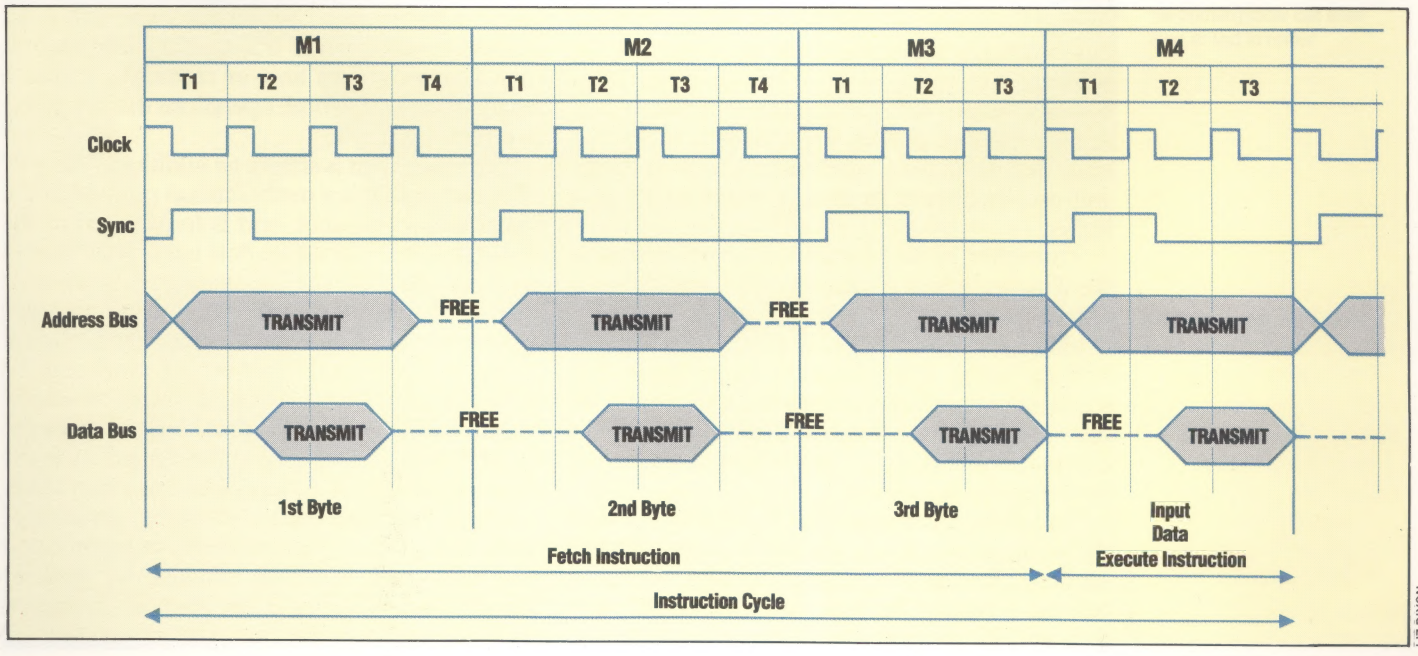


address $13FF to the accumulator'. This instruction would require three bytes: one to hold the binary code for the instruction ADD and two to hold the 16-bit address, $13FF. Let us say that it is stored in locations $1000, $1001 and $1002.

Before the instruction can be processed it must be fetched from memory. This requires three separate accesses to bring the three bytes along the data bus to the CPU. On completion of the fetch cycle, the complete instruction is in a special register within the CPU. All that remains is to decode the instruction and obey it. The instruction in our example requires a further memory access to get the contents of location $13FF so that it can be added to the accumulator.

All computer manufacturers publish the characteristics of their processors in the form of timing diagrams. These show the order of events for a number of different computer operations. We can draw up a timing diagram for the fetch and execute cycles of a machine code instruction. The timing of operations is controlled from the clock pulse (see page 246) and our graph shows that in this imaginary system the address bus is enabled by the sync pulse leading edge, while the data bus is enabled by the sync trailing edge. The sync pulse itself is triggered by the trailing edge of the first clock pulse in any operation phase, or machine cycle. The cycles are of different durations because the processor needs longer to decode the op-code byte of an instruction than to handle the operand bytes: the op-code must be decoded immediately because it specifies the number of operand bytes.

Fetch And Execute
A machine code instruction consisting of an op-code byte followed by two operand bytes is handled in an instruction cycle comprising fetch and execute phases. During the fetch phase the address bus accesses the memory locations holding the instruction, and the data bus carries the instruction bytes to the CPU. Here, the data and address buses are still busy during the execute phase because the instruction being executed causes a memory access



LIZ DIXON

CHAIN

This is the name for another form of data structure, similar to 'tree', 'stack' and 'list'. In a *chain,* each item of data contains a pointer to the location of the next item. This is particularly useful for storing information on disk, where for reasons of efficiency it is often necessary to spread a single file across several areas on the disk surface.

As the name implies, however, it needs only one of the links in the chain to be broken — perhaps by a tiny flaw in the magnetic recording surface — for the whole of the file to be lost, so additional means of access are usually built in for safety.

Many computers can also 'chain' programs. In this case the computer loads one program, runs it, then automatically loads another program and repeats the process. A number of short programs can thus be 'chained' to give the effect of a single long program. A common example of this is in games programs, which often load the playing instructions first, and, once they have been read, overwrite them with the main program. In this way the whole of memory becomes available to the main program and its variables, and no space need be wasted on storing instructions.

A *daisy-chain* is somewhat different in that it describes a hardware configuration -- usually the way that peripherals are connected to the central computer. One such configuration has each peripheral plugging in to its own dedicated port, another uses a common data bus to link all devices together (as in a local area network). In a daisy-chain, the first peripheral plugs into the computer's peripheral port. This peripheral features its own port, into which a second add-on device is plugged, and so on. The end result has the appearance of a daisy-chain, with the flowers representing the devices, and the stalks the links between them.

User-Defined Character
Character generators build text and other characters from a number of dots. Some character generators allow users to define their own characters, such as the Japanese symbol for 'bridge' shown here. However, few micros possess the necessary 16 by 16 grid of dots for such a complicated symbol

CHANNEL

A *channel* is a route through which data can flow, though in microcomputer terms it refers not to the bus or interface that transmits the data, but the software that controls it. With most operating systems, before data can be sent to a device (the screen, disks or printer, for example) a channel must first be opened, which amongst other things will usually reserve an area of RAM to act as a buffer.

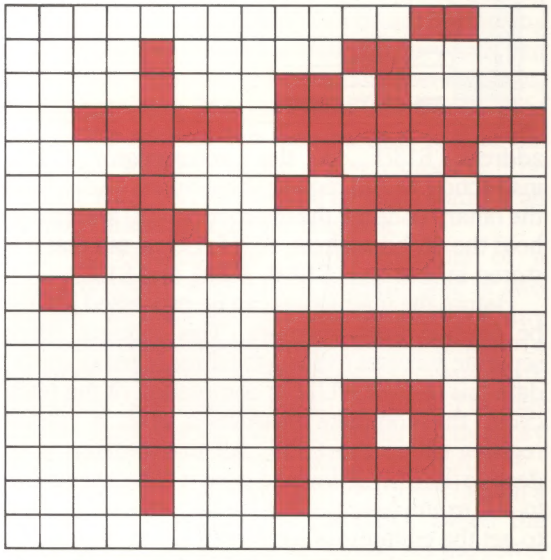
Thereafter, all data for that device will be sent to the channel, which is identified by a number or a name, and the operating system will then automatically route it to the hardware device, without further intervention by the program. On some older computers all this had to be set up by the programmer, whereas on many home computers the whole process is now taken care of by the operating system.

One of the advantages of using channels is that the programmer is not dealing directly with the hardware devices. If a program has been written to address a printer through, say, channel number five, and the programmer wishes to convert it to run on a plotter, he need only change one instruction at the start of the program to specify that channel five now refers to the plotter, instead of having to change all the PRINT and WRITE statements individually.

CHARACTER GENERATOR

The device that specifies the design of characters on the screen is known as the *character generator.* This is a section of the memory containing patterns of ones and zeros that specify the arrangement of dots on the screen.

These patterns are generally stored in ROM, although most home computers will allow the user to specify an area of RAM to replace the standard character generator. This enables you to alter the designs of any alphanumeric or graphic characters, and create your own customised character set.



LIZ DIXON

CHECK DIGIT/CHECK BIT

A specific number included in data to help spot errors in its recording or transmission is known as a *check digit* or *check bit.* Check digits are most commonly used when transmitting information over a long-distance line, or recording it onto a magnetic surface — both operations that are prone to errors.

The check digit is simply an additional piece of information that is a mathematical product of the actual data being sent, and is transmitted at the end of that data. When the data is received, or read back off the disk, the mathematical function is again performed, and if the new result does not agree with the check digit, an error has occurred.

The mathematical function involved may be quite complex, or it may be as simple as adding together all the bytes in the data block, dividing the result by 256 and then using the remainder as the check digit — this is usually known as a *checksum.*

Check digits are not the exclusive preserve of computers — they can be found in some credit card and cash retrieval systems, as well as International Standard Book Numbers found on book covers.

ORIC'S OFFSPRING

The Oric-1 was launched onto the UK market in 1983, but has never enjoyed the success of its competitor, the Sinclair Spectrum, because of design faults and a lack of software support. Now, Oric Products has launched a new and improved model, the Oric Atmos, which has remedied the shortcomings of the previous machine.

Equipped with a powerful Microsoft-style BASIC, a built-in Centronics printer port and a standard RGB monitor socket, the Oric-1 originally looked a good investment. However, a shortage of good software, coupled with some irritating bugs in the BASIC ROM, resulted in a lukewarm reception for the new machine.

Oric Products International has now rectified the major ROM errors and repackaged the computer as the Oric Atmos. The old calculator-style keyboard has been replaced by more professional full-travel typewriter keys, and the casing has been redesigned in a stylish red and black livery. The keyboard layout is the same as that of the Oric-1, with the addition of a Function key, which is as yet unconnected but is supplied in the interests of 'future expansion'.

The Atmos uses the 6502 microprocessor, and in normal operation has 37 Kbytes of RAM free for BASIC programming. Eight colours may be displayed by the Atmos, which has a maximum resolution of 240 × 200 pixels. The character set is held in RAM, allowing any character to be user-defined. There is also an alternative character set, which gives teletext-style block graphics. Unlike the Spectrum, which maintains a separate attribute file in RAM, the Atmos uses 'serial attributes'. These use less memory but are displayed on the screen as blank spaces, so care must be taken when planning screen displays. .

The Atmos ROM contains four pre-set sounds — ZAP, PING, SHOOT and EXPLODE — and these provide arcade-type sound effects. The MUSIC, PLAY and SOUND commands allow the user to take full advantage of the Oric's sophisticated sound chip, by setting a wide range of parameters to vary the sound. Volume ranges from very quiet to extremely loud, and the three tone channels and one noise channel give a seven-octave range.

The original Oric BASIC was notable for several annoying bugs. The TAB command did not work properly and the display was often corrupted by

IAN McKINNELL

Atmos System
The Oric Atmos is a modestly priced home computer with 48K of memory, colour graphics and sound. Oric makes two add-ons for the Atmos, both in matching colours. The disk drive gives a fast alternative to using a cassette recorder and the printer/plotter can draw lines or text in colour

Oric-1
The Atmos is an updated version of the Oric-1. It uses the same circuit board but has a different ROM chip that holds an improved version of BASIC. These changes are sufficient to make the Atmos a far better machine. Owners of the Oric-1 can pay £60 to upgrade to the Atmos standard. However, much of the Oric's software will not work on the Atmos, so users may find themselves unable to use their favourite programs

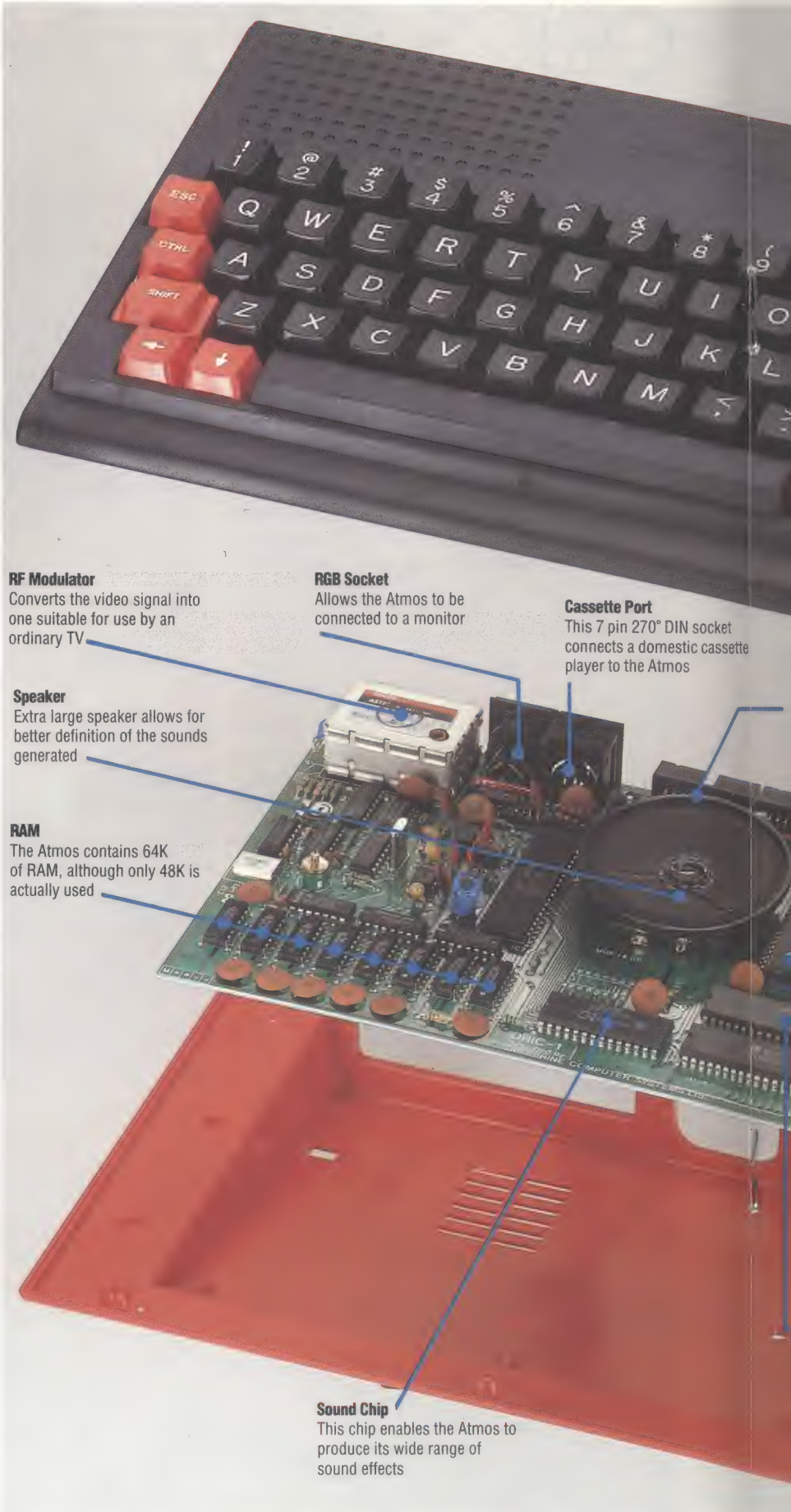
the various sound commands. The Oric also introduced spurious control codes when evaluating the STR$ function and gave incorrect results when LEN or VAL were used. The new ROM overcomes these difficulties. An unfortunate side effect of these improvements is the fact that Oric-1 machine code programs are unlikely to work on the Atmos, as several ROM routines have been relocated in memory.

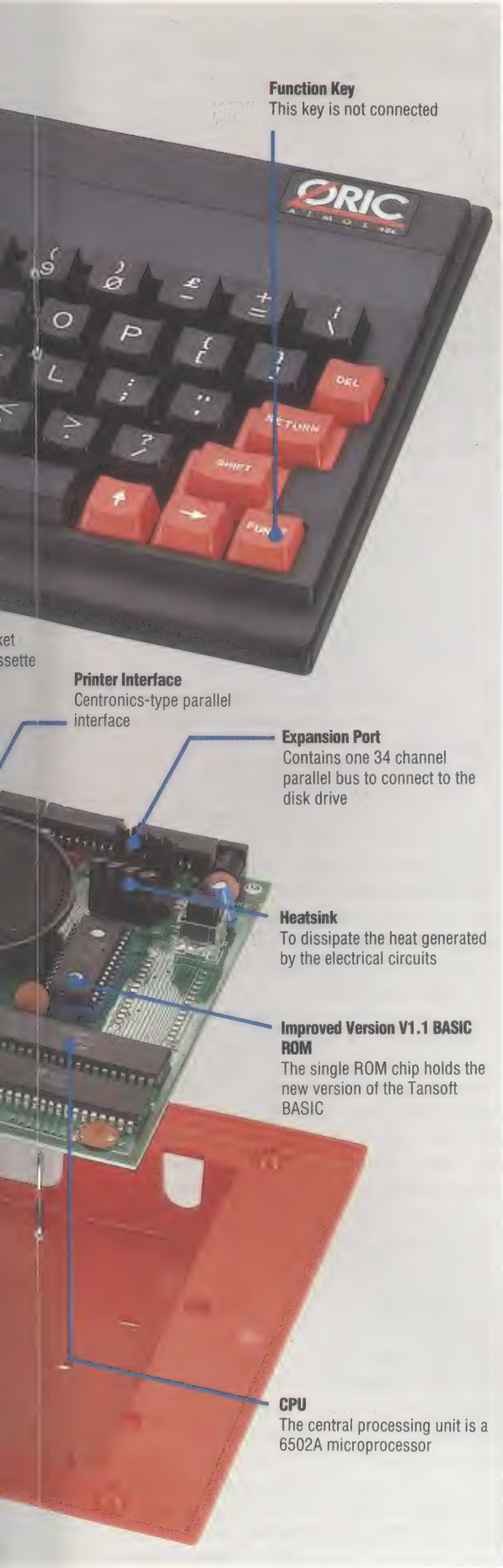
The BASIC is an extended version of the Microsoft dialect, developed by Tansoft from the original Tangerine BASIC. It supports the full IF...THEN...ELSE structure (Oric-1 BASIC had a bug in the ELSE segment of this command) and also provides the REPEAT...UNTIL loop instruction. An unusual feature is the provision of POP and PULL commands, which are used to jump out of GOSUB and REPEAT...UNTIL routines without producing an error report. Oric-1 BASIC would not allow the user to POKE an address with a hexadecimal value; this too has been corrected in the new ROM.

On early versions of the Atmos there were some problems with the new ROM. When designing the new chip, Oric included an updated error-checking routine for LOADing cassette tapes. This routine was so effective that users quickly discovered that the software found errors in programs on all but a very few cassette machines. However, Oric's redesigned ROM allows programs to LOAD satisfactorily.

To coincide with the new Atmos, Oric has redesigned its printer/plotter, which is now finished in the same red and black livery as the computer. Four small ballpoint pens (black, red, green and blue are the colours supplied with the unit) are set in a revolving plotting head; any colour may be selected under software control. The printer/plotter has a slow text speed of 12 characters per second but prints on plain paper.

The long-awaited microdisk drive has also been redesigned in the new Atmos colours. Oric has opted for the Hitachi 3" disks; these are encased in a rigid plastic shell. The Atmos can use up to four drives — a single master unit with an on-board disk





CHRIS STEVENS

Disk Drive
Oric's disk drive uses 3 inch disks. These come in a protective hard case. The drive gives a capacity of 160K on one side of a disk. The disks can be turned over and used on the other side to give a total of 320K per disk. The Oric disk drives can only cope with sequential files, rather than the more useful random access type

interface system, and up to three slave drives. As yet, no slave units have been produced, but these are expected shortly. The disk drive comes with a separate power transformer, which is powerful enough to drive two disk units and the computer itself. In early versions of the disk operating system, problems occurred when the printer and the disk drive were both swiched on. Any attempt to edit a program line resulted in the edited line — as well as all of the numbered program lines — being deleted from the listing. Oric claims that later versions of the operating system have overcome this difficulty.

Although early versions have been subject to problems in printer and cassette use, Oric Products seems to have given a great deal of thought to the production of the Atmos and its peripherals. The design team has taken note of criticisms levelled at the earlier Oric-1 and most of the errors have now been corrected. Oric-1 owners were poorly served by software manufacturers and, to counter this, Oric Products has commissioned Tansoft to produce a set of programs to be used with the disk drive. If software production increases, the Atmos should capture a larger share of a very competitive market.

Printer/Plotter
One excellent add-on for the Atmos is a printer/plotter. It uses four ball pens to draw lines and text in colour. It can draw text in sizes ranging from miniscule to several inches high. Its drawbacks include paper only $4\frac{1}{2}$ inches wide, slow speed and expensive ball pens. Because the printer/plotter works by drawing thin lines it is not suitable for drawing solid areas of colour

CHRIS STEVENS

ORIC ATMOS

PRICE
£169

DIMENSIONS
278×178×50 mm

CPU
6502

MEMORY
48 Kbytes RAM, 16 Kbytes ROM

SCREEN
26 rows of 40 columns in text mode and up to 200×240 pixels in eight colours

INTERFACES
An expansion port for a Centronics printer interface, cassette port and RGB socket

LANGUAGES AVAILABLE
Extended BASIC and FORTH

KEYBOARD
58 typewriter-style keys. The Function key is not connected

DOCUMENTATION
The manual is thorough and written in a pleasant conversational style that is clearly intended to take the beginner easily through BASIC programming. For the more advanced user there are chapters covering machine code and advanced input/output techniques, as well as a number of appendices giving full technical information

STRENGTHS
The Atmos has a wide range of facilities that are not available on more expensive machines. The BASIC is concise and contains a number of commands that make programming easier

WEAKNESSES
The method of screen display on the Atmos is hard to work with in high resolution mode. The disk drives are disappointing, being suitable for sequential access only

KEY POSITIONS

In the last instalment we introduced random access files and explained how they contrast with sequential files in terms of the amount of information they can handle and the speed at which this can be accessed. We now look at how this information can be organised in random access files, using the index and hash facilities.

If you have experimented with random access files, you will appreciate that they can make programming with files very easy. You can specify any particular record to read or write to, without having to bother with the cumbersome procedures needed to retrieve data stored sequentially. However, the methods of insertion and deletion that we detailed on page 244 are not the most efficient means of using random files; a far better method is to access the information using an index.

To create an index, a particular field of each record must be specified as a key. The value of this field will then be used to select particular records for display or processing. An index is thus built up, consisting of the value of the key field for each record, together with its corresponding record number. Thus if the record relating to Hilda Zeff is record number 17, and is the eighth record in an alphabetical listing, then the index array will store 17 in the eighth position. If the index is regularly sorted, searching for a record can be a very fast process.

The index is usually stored in RAM in order to be readily accessible. It can be generated on the spot, with a routine that reads through the entire random file, pulling out each key field into an array. This index can then be sorted ready for use. However, this introduces a long delay. An alternative method is to store index files on disk as well as data files. In this way, any number of key fields for a file can be created through index files stored on the disk. This will also enable the file to be indexed in different ways: for example, records could be given in name order (both A to Z or Z to A), date order, and so on.

How would you store an index file? An index file needs to contain two fields (the key data and the record number) for each record. This will be read into memory in full, ready for use and will be written out again only if it is to be updated or amended. This is an ideal application for a sequential as opposed to random access file, because the data is required in the order in which it is stored. This is one area where sequential and random access files complement each other.

Removing unwanted records from indexed random files is simply a matter of marking the records as deleted and ensuring that they no longer feature in the index. The most economical way to do this is to write a 'deleted' marker into the record — an asterisk at the start of the first field perhaps — as the whole field consisting of a delete flag would be a waste of space. The key for that record could then be removed from the index, or, alternatively, the record number could be set to some special value that indicated the record was deleted, −1 perhaps.

Whatever the method chosen, it is important that there is a record of the deleted records in the file. When new records are added to the file they can overwrite a deleted record. The original index entry must be replaced with a new one and, at a convenient point, the index re-sorted to include the new record at the correct position within the file. In this way, the program will provide a facility to recover accidentally deleted records, provided they have not been overwritten in the meantime.

It is a good idea to provide a way of tidying up the index file. The system of indexing we have detailed invites records to be stored out of order and with numerous and unnecessary gaps between them. While the file will work, the speed of access will gradually slow down. The tidy-up routine should sort the records into a convenient order and discard any deleted records within the file. Tidying up can be carried out as a user option or perhaps automatically whenever the system finishes a major operation.

Indexing is not the only means of finding specific records in a large file very quickly. *Hashing* is an alternative method that is well suited to extremely large files and is therefore usually only seen in hard disk systems, or machines with very high capacity floppy disks. However, many operating systems and programs use hashing internally to speed up their operation, so it's certainly a technique worth knowing about.

Hashing replaces the index with a formula or hashing algorithm. This takes the value of the key field and produces from it a record number, known as a hash. The record that goes with the key is then stored at this position in the file. The formula would be devised according to the type of data in the key field. If the key field contains a date — to enable the records to be sorted chronologically — you might use the month number, multiplied by the last two digits of the year, plus the day number. A name field could be hashed by manipulating the ASCII codes used for the letters within the name and so on.

Suppose we want to create a hashed file of

employee records using surnames as the sort key. The hashing algorithm that we will use is: take the ASCII codes of the first four letters and treat them as an eight-digit number, square that number, then take the last four digits of the number as the hash. JONES, therefore, hashes into record 1161, whereas JONQUIL hashes into 0161.

Hashing is very different from an indexed system. With hashing, you can only have one key field (and one hashing algorithm) per file and this is used when first placing the records in the file. Any number of indices can be associated with a particular file and these can be created at any time after or during the file's creation.

Hashing is less flexible than indexing but it is much quicker. To find a particular record, the program just takes the key, hashes it and retrieves that particular record. The time taken to search an index (and indeed to create it in the first place) is therefore dispensed with.

A problem with hashing arises when two records generate the same hash code and therefore should occupy the same position in a file. To avoid this, hashing algorithms are carefully designed so that no two keys (save for identical ones) generate the same hash. Additionally, records are spaced out in the file so that two hashes that are apparently next to each other actually cover a gap of five or so unused records.

We can now clarify our description of a hashing system as follows. When a record is stored, its key is hashed to produce a record number. If that record is occupied, the system looks at the next record sequentially. It can do this for the whole block of five (or whatever) records associated with that hash. When a record is to be retrieved its key is hashed and that group of records is then searched sequentially for an exact match. This may seem to nullify the speed advantage, but what hashing effectively does is to reduce the number of records to look through from perhaps three thousand to five or six.

What happens if all five or so records for a particular hash become filled? There are several ways to cope with this, the obvious one being to report a 'file full' message. More often, records that can't be fitted in position in the file are written to a separate overflow file with its own index and incorporated into the main file when possible. Most systems make a determined effort to avoid overflow by habitually keeping hashed files only 80 per cent or less full. This highlights another limitation of hashed access to random files. A hashed file tends to consume more space than if the system used an index.

Hashing also speeds up the deletion of unwanted records. You simply hash the key of the record, do a quick search to locate it exactly and mark its position as unfilled. It will then be overwritten the next time a record with an identical hash is added to the file.

In the final instalment of this series we will look at the BASIC commands necessary to create and access cassette files.

Index Linked

Find 'Davids'

Key	No.
Andrews	1
Baker	–1
Brown	5
Cressy	–1
Davids	7
Dawes	23
Fish	15
Gregory	28
Haynes	37
Johns	25
Klaus	11
Marks	10

Index File

The most common way to access a random file is with an index. This is a list in RAM showing the values for a particular key field with the corresponding records. When a record is being accessed, it can be quickly looked up in the index and read in to memory.

Deleted records are left in the file and marked as unwanted. They are then overwritten as new records are added

Main File

	Name	Work Tel.	Home Tel.	Job Title
1	Andrews	242 0791	727 0942	Designer
2	Phillips	636 2418	221 3940	Accountant
3	Smith	631 0836	286 8170	Editor
4		Deleted record		
5	Brown	729 8213	236 2190	Dentist
6	Peter	836 6622	298 4310	Decorator
7	Davids	743 7216	450 6926	Gardener
8		Deleted record		
9		Deleted record		
10	Marks	730 6321	429 7592	Mechanic
11	Klaus	493 9899	455 8431	Lawyer
12	West	736 7700	693 0452	Hairdresser

Making A Hash

Find 'Davids'

HASHING ALGORITHM

The hashing algorithm converts the keys so that it refers to a particular block of records
Records with an identical hash are grouped together

Unused space between blocks of records is left so that new records can be inserted into position

Name	Work Tel.	Home Tel.	Job Title
Davidson	629 0491	430 0592	Plumber
Day	436 2488	362 0066	Director
Darran	730 0021	626 9191	Cleaner
Dammat	439 9933	630 4918	Writer
Davids	743 7216	450 6926	Gardener
Dawes	830 0123	340 9924	Nurse
Egerton	731 6666	458 0021	Designer
East	831 8294	450 6218	Caterer

Hashed files offer high-speed access to particular records in large random files. However, the system is restrictive and needs careful programming.

The record key is processed into a position in the file with a predetermined hashing algorithm. Each possible hash usually refers to a block of records that can be searched sequentially to find the required record

KEVIN JONES

NOW HEAR THIS

```
  10 REM******AUDIO GAME******
***
  20 PRINT"A SIMPLE AUDIO GAM
E":PRINT:PRINT"MOVE YOUR SPACE
SHIP TO THE BEACON BEFORE
    YOUR FUEL RUNS OUT"
  30 PRINT:PRINT"     THE NEA
RER YOU ARE, THE HIGHER
       THE BEACON'S PITCH"

  40 PRINT:PRINT"           C
ONTROLS ARE:":PRINT" I (UP)  M
 (DOWN)  J (LEFT)  K (RIGHT)"
  60 PRINT:PRINT"  *******PRE
SS ANY KEY TO START*******"
  70 A$=INKEY$(0):IF A$="" TH
EN GOTO 70
  80 P=INT(RND(1)*500)+1:Q=IN
T(RND(1)*500+1):F=3*(P+Q)/2:X=
0:Y=0
  90 XD=0:YD=0
 100 PRINT "FUEL=";F," DISTAN
CE=";
 110 IF (ABS(P-X)<2 AND ABS(Q
-Y)<2) THEN PRINT"YOU MADE IT
 - WITH ";F;" FUEL UNITS LEFT":
END
 120 A$=INKEY$(0):IF A$="" TH
EN GOTO 120
 130 IF (A$="I" AND YD<3) THE
N YD=YD+1
 140 IF (A$="M" AND YD>-3) TH
EN YD=YD-1
 150 IF (A$="J" AND XD>-3) TH
EN XD=XD-1
 160 IF (A$="K" AND XD<3) THE
N XD=XD+1
 170 X=X+XD:Y=Y+YD
 180 D=SQR((P-X)*(P-X)+(Q-Y)*
(Q-Y))
 190 PRINT D
 200 SOUND 1,-8,(255-D),2
 210 F=F-ABS(XD)-ABS(YD):IF F
>0 THEN GOTO 90
 230 PRINT TAB(10);"*******CR
ASH*******":PRINT TAB(11);"***
OUT OF FUEL***"
 240 PRINT:PRINT"POSITION IS
";D;" SPACE UNITS FROM BASE"
 260 END
```

The most popular computer games rely heavily on the exciting visual effects that can be produced by home micros. While many games make a lot of noise, few of them make really creative use of sound. We continue our DIY programming series by suggesting a few ideas for the development of interesting and enjoyable 'audio games'.

Arcade games use sound to support their well-developed graphics effects. Although sound effects are used to heighten the realism and excitement of the action, they are rarely the focus of the game itself. However, if we can use our visual sense as the basis for so many varied computer games, there is no reason why we can't generate games that are centred on our sense of sound.

The game we give here is such an 'audio game'. While we do not claim that it is the most exciting game ever developed, it is certainly interesting to play. What seems to be quite a simple task at first soon becomes a fascinating challenge simply because you have to use a different sensory input than that normally used when controlling some brightly coloured screen display.

In our game you are placed at the controls of a spacecraft that is running short of fuel. Your only hope of survival is to dock with a fuel station nearby. Unfortunately, owing to a malfunction in your computer system, you have no visual information to guide you in your efforts at docking; the only method of navigation is to home in on the station using an audible signal. The pitch of the 'navigation beacon' signal becomes higher the nearer you get; so it's just a question of listening carefully and responding precisely with the controls.

Just like a real spacecraft, once you set your ship off in a particular direction, it will keep going until you counteract that motion by using the opposite control. If you use two Us to move up quickly, then you'll need to press two Ds to stop again. This makes the game much harder than it seems.

It will take some practice for you to be able to complete the game using only the sound cues. By adding a few simple lines, however, you could give an indication on the screen of where the ship is in relation to the fuel station. This could simply show the distance between you and the station (the variable D) or it could tell the player the most effective buttons to push (if SGN(p-x)= -1 then you need to go left, for example). These helpful hints should make the game much easier to play.

Having tried our program, you may like to create your own games. Using sound to locate or avoid objects is a field with considerable programming possibilities. How about sonar games for submarines or a minefield that you have to navigate using a detector that emits an audible warning? A safe cracking game shouldn't prove too difficult.

The more advanced sound capabilities of machines like the BBC Micro and the Oric-1 will obviously be an advantage here. Sound games can use several voices simultaneously, or give them slightly different noises, each with its own significance in the game. The programmable volume level available on some machines will be a useful feature.

DAVE COOPER-SMITH

Basic Flavours

This program was written on a BBC Micro in Mode 7, and is, therefore, almost standard Microsoft BASIC. Its PRINT commands presuppose a 40-column screen display, and will need formatting for different displays. The SOUND command in line 200 is specific to BBC BASIC. The value of the parameter (255-D) is the pitch of the note to be played, while the other parameters control volume, duration and channel. INKEY$(0) in line 120, and the use of RND in line 80 will need attention on other machines:

Spectrum

Insert LET in all assignment statements. Change INKEY$(0) to INKEY$. Change RND(1) to RND. Change line 200 to:

200 BEEP 0.4,(255-D)

and insert:

15 RANDOMIZE
195 IF D>254 THEN LET D=D-254

Commodore 64/Vic-20

Change INKEY$(0) to GET A$. Refer to your user manual for Commodore's sound commands. Insert:

75 X=RND(-TI)

Dragon

Change INKEY$(0) to INKEY$. Change RND(1) to RND(0). Change line 200 to:

200 SOUND (255-D),10

and insert:

195 IF D>254 THEN D=D-254

Oric Atmos

Change INKEY$(0) to KEY$. Change line 200 to:

200 SOUND 1,(255-D),9:WAIT 40:PLAY0,0,0,0

and insert:

195 IF D>254 THEN D=D-254

WATER SPORT

With the software market currently dominated by space invaders-type 'shoot-'em-up' games, it is refreshing to find a new game that shuns the well trodden path and heads off into virgin territory. Durell Software's Scuba Dive is such a game: an underwater adventure that has soared high in the software pop charts.

There are three versions of Scuba Dive available: one for the Spectrum (priced at £5.95) written by Mike Richardson; one for the Oric-1 (£6.95) by Ron Jeffs; and another for the Commodore 64 (£6.95) by Nigel Dewdney. The Oric-1 version is being adapted to run on the Oric Atmos as well.

The player takes the role of a scuba diver collecting treasure from the seabed, and risking life and limb to do so. Our intrepid hero's main objective is to collect points-accumulating pearls, which are taken from oyster shells and giant clams. At a more advanced stage of the game you must collect treasure from chests deep within the cavernous seaworld.

However, there are many good reasons for exercising caution as you go. The water is heavily populated with creatures that have a detrimental effect on scuba divers: jelly fish, octopuses, squid, electric eels and, in the Spectrum version, sharks! If you touch any of these creatures you lose a life, although none of them will attack you on purpose. You must simply avoid them at all costs. Another danger is discovered when you start retrieving pearls from the giant clams. These have the ability to slam shut on you, trapping you in their grasp.

The narrow entrances to the main cavern and the lower-level depths are guarded by giant octopuses. Passing these can be tricky, as their tentacles are constantly waving about. But every now and again you can manage to sneak past. The Commodore version is octopus-less. Instead, a trap door bars your way. This is constantly opening and shutting and you have to slip through without being knocked unconscious.

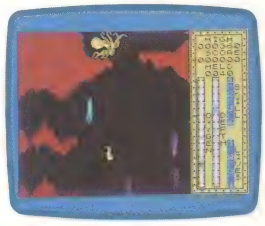
The program allows you three lives per game, and has a difficulty gradient from one to four. A life is lost if you touch any of the aquatic wildlife or if you run out of oxygen.

Each version of the game starts off with a view of the surface of the sea and a large portion of the depths. The boat from which you dive is bobbing on the surface. It is possible to get caught under the boat as you dive, so care is needed from the moment you go down. On the Spectrum version it is an advantage to have a good sense of direction because your boat can drift when you are underwater. In the Oric version this isn't so much of a problem, as the boat always moves from left to right in a screen 'wrap-around' fashion, and thus is never out of sight when you surface.

The quality of the graphics on the Spectrum version (which is by far the best of the three) is superb. Good use has been made of colour, and the creature and cavern design is realistic. Control over the diver is achieved by using the X and Z keys to turn clockwise and anti-clockwise, and the Space and Shift keys move the diver forward. Spectrum owners can also use joysticks, although the program will not work with the Kempston joystick interface. Commodore owners also have the choice to use joysticks, but Oric players must make do with the keyboard.

The Oric version is far less exciting than its Spectrum counterpart in several respects. Movement of the diver and the creatures is very jerky, and the graphics — especially the cavern walls and the chests — are far less detailed. The Commodore version also lacks the Spectrum's detail, but it performs far more satisfactorily than the Oric version.

Quality Control
These screen pictures illustrate the difference in quality between versions of the Scuba Dive game



Sinclair Spectrum

Oric-1

Commodore 64

Seen In Perspective

This image is composed of several screen dumps of the Spectrum version of Scuba Dive. The images are joined together to show the diver's view of the underwater caves

SCREEN SHOTS BY IAN McKINNELL

REGISTERED ADDRESS

So far in the course we have taken a detailed look at how the CPU manipulates memory, using registers such as the accumulator and ALU. Now we can begin to look more closely at how simple procedures are performed in machine code. Here, we concentrate on the basic arithmetical operations of addition and subtraction.

The differences in operation between the Z80 and 6502 microprocessors in the way they go about performing these simple arithmetical tasks reveal the different philosophies behind their design. The Z80's many registers, with their sophisticated set of operation instructions, typify the processor itself — elegant, complex and powerful. The much simpler 6502 architecture and operation set seem to suggest an altogether humbler processor, which is robust and practical but apparently not quite in the Z80 class. This impression is accurate as far as it goes, but the 6502's wealth of addressing modes and its use of zero page as an extra index register, give it a subtlety and versatility that will enable it to dominate the home and business micro world for some time to come.

The great advantage of the Z80's registers is their flexibility — they can be treated simultaneously as both two-byte or single-byte registers, thus allowing enormous addressing scope. The 6502, on the other hand, has no two-byte registers, but is able — by way of its addressing modes — to treat zero page as an array of single-byte and two-byte registers.

ARITHMETICAL BASICS

We have seen that the CPU registers permit a variety of possible memory accesses, but manipulating memory usually requires something more than simply loading, storing and comparing its contents. The ability to perform the four operations of arithmetic is essential to a computer system, yet both the Z80 and the 6502 support only addition and subtraction. Multiplication and division must be programmed, as must the addition and subtraction of numbers larger than $FF. This is a limitation of both of the CPUs, though the educational value to the programmer of having to invent multiplication and division algorithms is enormous. On the 16-bit processors that succeeded the Z80 and 6502, however, both operations are supported, thanks to the greater speed and power of the CPUs.

We have used the ADC ('add with carry') instruction and a variety of INC ('increment') instructions, in doing single-byte arithmetic on both CPUs. Here are the two ways of adding the contents of two two-byte memory locations:

6502			Z80		
ADDR1	DW	$7E60	ADDR1	DW	$7E60
ADDR2	DW	$4A51	ADDR2	DW	$4A51
SUM	DS	$03	SUM	DS	$03
BEGIN	CLC		BEGIN	LD	A,$00
	LDA	ADDR1		AND	A
	ADC	ADDR2		LD	HL,(ADDR1)
	STA	SUM		LD	DE,(ADDR2)
	LDA	ADDR1+1		ADD	HL,DE
	ADC	ADDR2+1		LD	(SUM),HL
	STA	SUM+1		ADC	A,$00
	LDA	$00		LD	(SUM+2),A
	ADC	$00		RET	
	STA	SUM+2			
	RTS				

The single-byte method employed on the 6502 can be used on the Z80, but the register-pair method used in the Z80 version has no 6502 equivalent. Notice the strategies used to handle the various carry possibilities, starting with the CLC (6502) and AND A (Z80) instructions that clear the carry flag prior to the addition, and ending with the modification of the third byte of SUM. Allowing for the maximum result is vital in all arithmetic.

Subtraction can be treated similarly to addition, both processors having a SBC ('subtract with carry') instruction although two-byte subtraction is supported on the Z80. Because of the possibility of generating a negative result in subtraction, however, we must now begin to investigate the binary representation of algebraic sign.

To start, we need say no more about negative numbers than is implied by this statement:

If A+B = 0 then it follows that A= −B

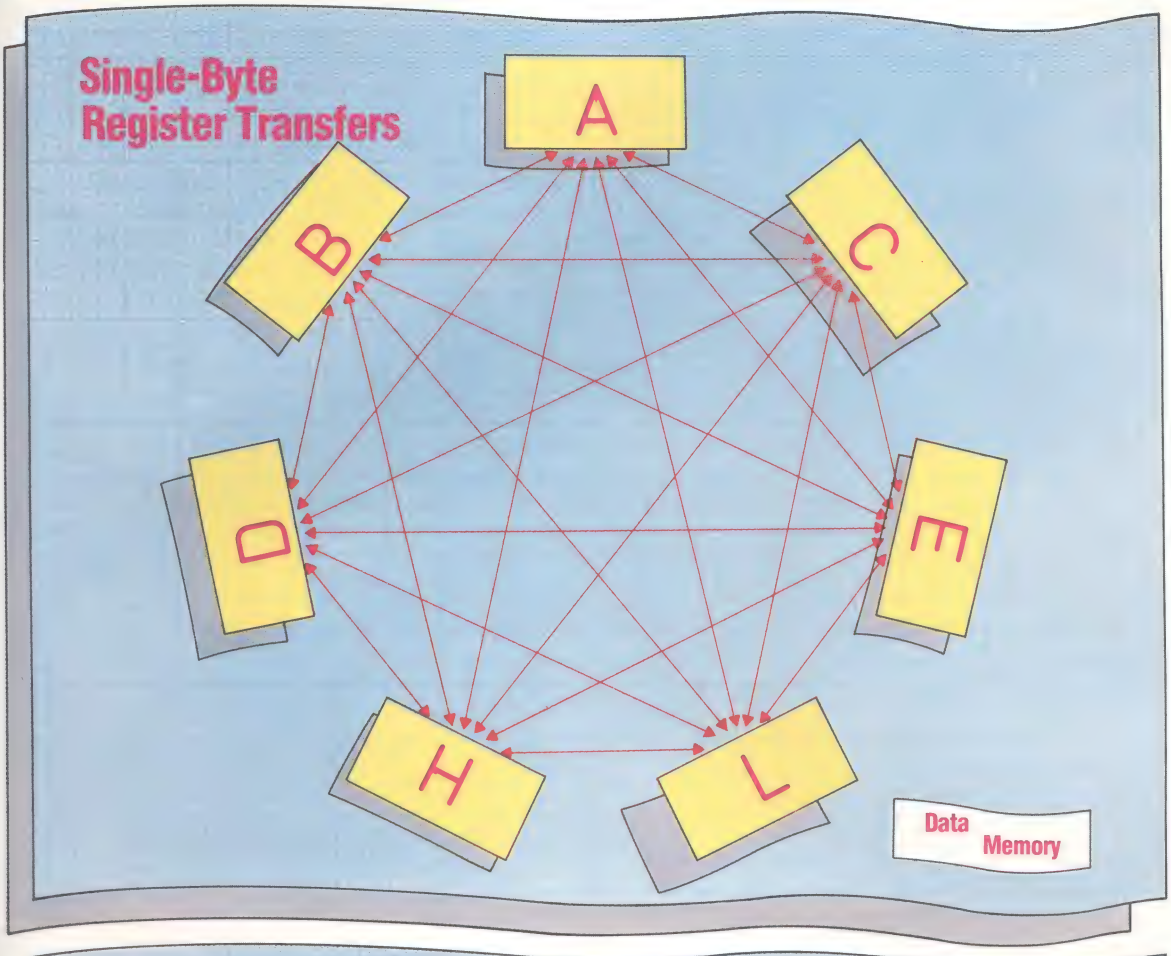
which implies that if A is a positive number, then its negation or complement is the number which when added to A gives a result of zero. For example, if A is the single-byte number $04, then its single-byte complement is $FC:

$04+$FC=$100

Remembering that $100=$00 (if we have only a single-byte register for holding the result), this complementary representation means that subtraction can be seen as addition with negative numbers. That is:

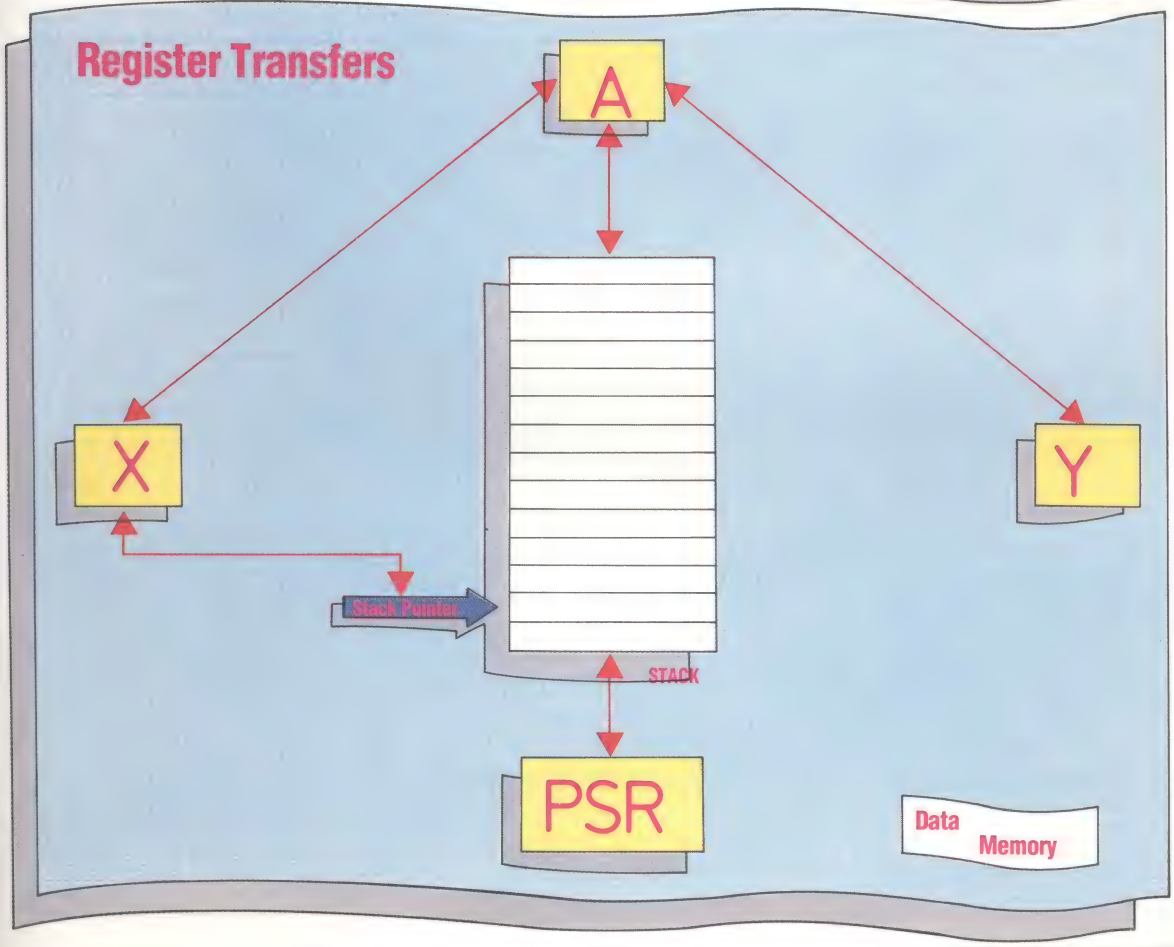
A−B is the same as A+(−B)

Thus, $08−$05 is the same as $08+(−$05), and (−$05)=$FB (as $FB+$05=$100), which means that our original subtraction problem can be re-



KEVIN JONES

Double Identity
The Z80's data registers can communicate as single-byte registers with every other single-byte register. They can each communicate with memory in direct, immediate, indirect, absolute, and indexed modes. When treated as BC, DE, HL – the two-byte register pairs – they can transfer 16-bit data to and from memory and the stack, and are effectively 16-bit accumulators for addition and subtraction. This combination of flexibility and resourcefulness is the key to the Z80's huge success



KEVIN JONES

Plain And Simple
The 6502's internal communication is severely linear, and restricted to eight-bit data transfers. Only the accumulator can communicate directly with X and Y; only X can communicate with the stack pointer; and only the PSR and the accumulator access the stack. Memory transfers are possible in absolute, direct, indirect, indexed, immediate, and zero page modes. The 6502's inventive use of zero page mode compensates for the small size of its register set; zero page can be treated as 128 two-byte CPU registers

expressed as $08+$FB. The result of this sum is $103, which is $03 as a single-byte number.

This kind of representation is known as *two's complement*: the complement of a single-byte number is formed by subtracting it from $100. There is another representation known as *one's complement*, and the two are related in an interesting way. Consider this:

```
$05 = 00000101      binary
$FA = 11111010      one's complement
           +1
-------------------
$FB = 11111011      two's complement
-------------------
$05+$FA=$FF
$05+$FB=$00
```

The one's complement of a single-byte number is formed simply by complementing or negating each binary bit of the number. If one is added to this result, then the two's complement of the number is produced. A number and its one's complement always total $FF, while a number and its two's complement always total $00 (actually $100). It is conventional then, in signed integer arithmetic, to regard the numbers from $00 to $7F as the positive numbers, (0 to 127) and $80 to $FF as the negative numbers (−128 to −1). If you compare the binary representations of these numbers you will notice that all the negative integers have bit 7 set, while in the positive numbers bit 7 is always reset. Accordingly, bit 7 is known as the *sign bit* of a signed number, and the carry flag of the processor status register is set or reset as a copy of bit 7 of the result of the last arithmetic or logical operation.

There is no easy way round this potentially confusing subject, and it simply has to be approached when you start doing signed arithmetic. Fortunately, once its implications are understood, it can be handled mechanically by rule-of-thumb methods. These methods, and the multiplication and division alogrithms, are the subject of the next instalment of the course.

Answers To Exercises On Page 259

1) The following program reverses the order of the character string stored at LABL1:

6502

```
;
ORIGIN  ORG   $7000
LAST1   EQU   $0D
LABL1   DB    'THIS IS A MESSAGE'
TERMN8  DB    LAST1
;
BEGIN   LDX   #$FF
        LDA   #LAST1
        PHA
LOOP0   INX
        LDA   LABL1,X
        PHA
        CMP   #LAST1
ENDLP0  BNE   LOOP0
CLRSTK  PLA
;
BEGIN1  LDX   #$FF
LOOP1   INX
        PLA
        STA   LABL1,X
        CMP   #LAST1
ENDLP1  BNE   LOOP1
        RTS
```

In the 6502 version, the code between LOOP0 and ENDLOOP0 uses X-indexed addressing in a loop to load the characters one-by-one from LABL1, and push them onto the stack — having first pushed the ASCII value of the terminator character to mark the bottom of the stack. The last character pushed onto the stack is also the terminator, this time determined from its position as the last character in the string. This concludes the loop, and the terminate character on top of the stack is then cleared at CLRSTK.

The Z80 version uses IX in indirect addressing mode to load the accumulator from LABL1 onwards, and pushes not only the accumulator but also the flag register onto the stack. This means that the characters of the string at LABL1 are interspersed on the stack with successive values of the processor status register.

Z80

```
        ORG   $C000
LAST1   EQU   $0D
LABL1   DB    'THIS IS A MESSAGE'
TERMN8  DB    LAST1
;
BEGIN   LD    IX,LABL1-1
        LD    A,LAST1
        PUSH  AF
LOOP0   INC   IX
        LD    A,(IX+0)
        PUSH  AF
        CP    LAST1
ENDLP0  JR    NZ,LOOP0
CLRSTK  POP   AF
;
BEGIN1  LD    IX,LABL1-1
LOOP1   INC   IX
        POP   AF
        LD    (IX+0),A
        CP    LAST1
ENDLP1  JR    NZ,LOOP1
        RET
```

The code between BEGIN1 and ENDLP1 in both versions is a reflection of the previous loop and uses the same techniques, but this time pulling the character string off the stack in reverse order, and storing it at LABL1 onwards. The loop finishes when the terminator character is found at the bottom of the stack.

Notice how important it is to balance stack pushes and pulls, and that the most difficult part of the problem is deciding how to handle the extreme conditions — what to do at the start of the loops, how to terminate them, and what 'tidying-up' (if any) is then required.

The Z80 instruction at BEGIN and BEGIN1 (LD IX,LABL1−1) illustrates the usefulness of an assembler program. Here, it decodes the expression (LABL1 −1) to mean 'the address of the byte immediately before the byte whose address is LABL1', and assembles that address into the code. Most assemblers support some measure of expression evaluation, usually allowing one or two operands to be modified by a single arithmetic operator — normally '+' or '−'.

2) This program reverses the order of characters in each word of the string at LABL1, while maintaining the order of the words themselves:

6502

```
;
ORIGIN   ORG   $7000
LAST1    EQU   $0D
SPACE    EQU   $20
LABL1    DB    'THIS IS'
TERMN8   DB    LAST1
;
BEGIN    LDX   #$FF
LOOP0    JSR   RVSWRD
         CMP   #LAST1
ENDLP0   BNE   LOOP0
         RTS
;
;****REVERSE A WORD S/R****
LASTCH   DB    $00
LASTX    DB    $00
RVSWRD   TXA
         TAY
         INY
RVSLP0   INX
         LDA   LABL1,X
         PHA
         CMP   #SPACE
         BEQ   CLRSTK
         CMP   #LAST1
ENDRV0   BNE   RVSLP0
CLRSTK   PLA
         STA   LASTCH
         STX   LASTX
RVSLP1   PLA
         STA   LABL1,Y
         INY
         CPY   LASTX
ENDLP1   BNE   RVSLP1
         LDA   LASTCH
         RTS
```

There are several points of interest here: the use of JSR and CALL instructions, for example. The RVSWRD subroutine is similar in structure to the program given in Exercise 1, but it reverses only the characters of a word, not the whole string. In both the 6502 and Z80 versions, the index register (X and IX respectively) is used to pass the start address of the word to the subroutine, and the accumulator is used to pass back to the calling program the value of the character that terminated the work (either a space or the string terminator character). Passing values this way is a very common Assembly language technique, and must be used with care — especially if you are in the habit of pushing all CPU registers at the start of every subroutine (as demonstrated on page 258).

Another significant feature is the use of the Y register in the 6502 version, first to hold the start address of the word while X is used as an index on the stacking loop, then as an index on the 'un-stacking' loop while X holds the end address of the word. 'Address' is used imprecisely here as X and Y are single-byte registers, so neither can hold a full address. Instead, in this case they hold an offset to the address LABL1. In contrast, the Z80 IX and IY index registers can hold a full two-byte address.

In the Z80 version, IX and IY are not used at all — the HL and DE register pairs are used instead. Like the 6502 X and Y registers, these hold the word start and

Z80

```
         ORG   $C000
LAST1    EQU   $0D
SPACE    EQU   $20
LABL1    DB    'THIS IS A MESSAGE'
TERMN8   DB    LAST1
;
BEGIN    LD    DE,LABL1-1
LOOP0    CALL  RVSWRD
         CP    LAST1
ENDLP0   JR    NZ,LOOP0
         RET
;
;***REVERSE A WORD S/R***
LASTCH   DB    $00
RVSWRD   PUSH  DE
         POP   HL
         INC   HL
RVSLP0   INC   DE
         LD    A,(DE)
         PUSH  AF
         CP    SPACE
         JR    Z,CLRSTK
         CP    LAST1
ENDRV0   JR    NZ,RVSLP0
CLRSTK   POP   AF
         LD    (LASTCH),A
;
RVSLP1   POP   AF
         LD    (HL),A
         INC   HL
         LD    A,L
         CP    E
         JR    NZ,RVSLP1
         LD    A,H
         CP    D
ENDRV1   JR    NZ,RVSLP1
         LD    A,(LASTCH)
         RET
```

end addresses, but instead of being indexes on a base address, they are used as indirect addresses (the instruction LD A,(DE) means 'load the accumulator from the byte whose address is held in DE'). All the Z80 register pairs can be used in this way. An odd limitation of the instruction set is the lack of any two-byte comparison instruction. Thus, comparing the contents of DE and HL involves comparing E with L, then D with H. Similarly, in the 6502 version, X and Y are compared indirectly using a memory location, since there is no instruction for comparing X with Y.

QUICK SELLERS

Just about every week a new software house specialising in games programs appears in the market place. Yet very few of these ever seem to establish a permanent place for their products on the high street shelves. One of the exceptions is Quicksilva, which since its inception in 1981 has rarely been out of the top ten best sellers lists.

The production of the Sinclair ZX80 microcomputer was the initial impetus for Quicksilva. The success of the machine encouraged a self-employed test engineer called Nick Lambert to design a three Kbyte add-on board to supplement the ZX80's meagre one Kbyte memory. This he sold successfully by mail order. When Sinclair released the ZX81 the following year, Lambert formed Quicksilva with John Hollis and Mark Eyles to produce a series of add-ons for the new machine. 1981 also saw the appearance of Defender, a game written by Lambert, which was Quicksilva's first venture into the software market.

The success of its Defender game encouraged Quicksilva to produce other arcade-style software, and the hardware side of the business was eventually dropped. In April 1982, Quicksilva was launched as a limited company.

The company's second big software success was an adventure game called Timegate. By Christmas 1982, there were 10 Quicksilva games on the market, and in January of the following year, W H Smith ordered 10,000 copies of Timegate. With its products — and its name — penetrating the rapidly expanding high street software market, the demand for Quicksilva's products soon rocketed. Having been launched on an overdraft of £200, the company's turnover for its first year of operation amounted to £70,000. Quicksilva products are now stocked by the large chain stores and by 150 independent retailers. The company also believes that its games reach over 70 per cent of the world market.

Such a considerable upsurge in demand has meant that the company has had to expand quickly beyond a three-man operation. Quicksilva now advertises for software authors in the computer press, and a games writer can expect to earn up to 15 per cent royalty on every one of his cassettes sold. The company today continues to diversify and has moved away from concentrating entirely on the Sinclair machines: the current Quicksilva catalogue includes games designed for the BBC Micro, the Dragon, the Commodore 64 and the Vic-20.

Quicksilva is now regarded as one of the major software publishing houses in the UK. Its name was further enhanced by the vast success of Ant Attack (see page 6), a game that featured extremely sophisticated graphics. Rod Cousens, who took over as managing director when Lambert decided to concentrate on more creative projects, was elected vice-chairman of the Guild of Software Houses and 'Person of the Year' by the Computer Trade Association in 1983.

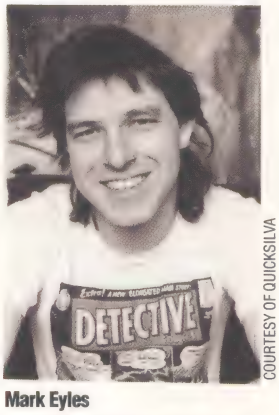
Quicksilva continues to look for unexpected directions in which to diversify. In particular, the company has recently released a 'non-violent' game called The Snowman, which is based on the popular children's story by Raymond Briggs. It is regarded as a welcome antidote to the vast plethora of 'zap-'em' space invaders type of game. In other ways, the company's plans are a little more predictable: soon it hopes to make a big splash in the North American software market.

MAP BY KEVIN JONES COURTESY OF YOUR SPECTRUM

On Offer
These are some of the latest games on offer from Quicksilva. The company now produces games for a wide range of computers

IAN McKINNELL

Rod Cousens
The present managing director of Quicksilva



Mark Eyles
Advertising director and one of the founders of the company

COURTESY OF QUICKSILVA

DATA IN STORE

Coming soon in THE HOME COMPUTER ADVANCED COURSE

Making Waves

Beautiful patterns can be created with ease by using the right geometrical formulae. We show you how with a program for the BBC Micro, Spectrum and Oric–1/Atmos in **Issue 17**

Travelling Light

Computer buffs can take their enthusiasm travelling and impress their fellow train passengers if they equip themselves with a lap-held portable micro. Our feature in **Issue 18** reviews half a dozen of these light-weight wonders

Flower Power

The choice of the right printer can be one of the most confusing topics that the computer user has to face. In **Issue 19** we shed light on daisy wheel printers, which offer the highest quality of print – but have snags, too

Tread Lightly

Minefield is a favourite computer game calling for quick wits and a fast finger on the keyboard. In a new series beginning in **Issue 20**, we program this game for the BBC Micro, taking advantage of the machine's superb graphics capability

Power To The People

If computing leaves you with no money to invest on the stock exchange, do the next best thing by playing a business game on your micro. We review a portfolio of these simulations in **Issue 21**

Trace Elements

Give your computer some pictures to look at by linking it to a digitiser – a device that turns images into numbers. Digitisers are explained and compared in **Issue 21**

All this and, of course, the *Home Computer Advanced Course* regular features: applications, reviews of hardware and software, programming techniques, machine code programs, and much more…

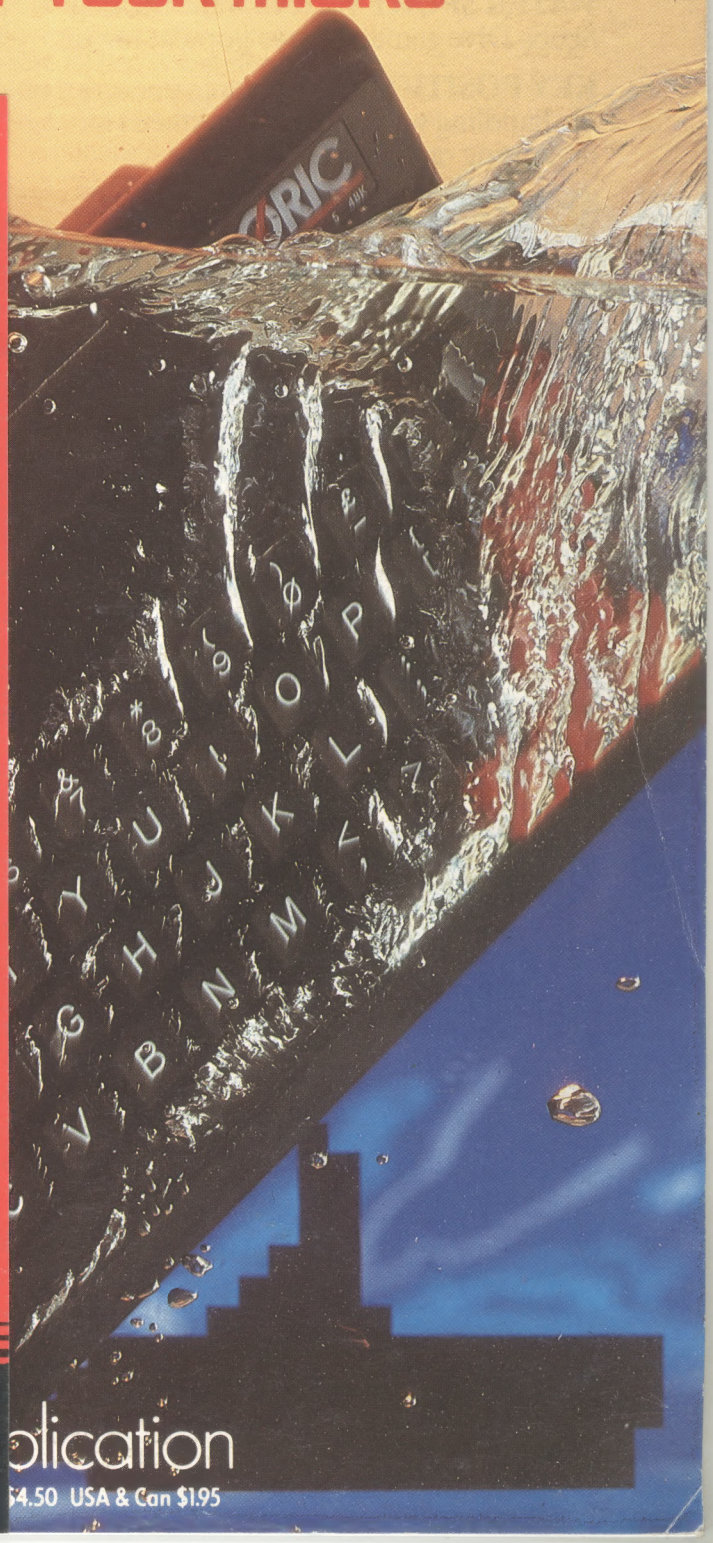
PLACE A REGULAR ORDER WITH YOUR NEWSAGENT

ORIC

ISSN 0265-2919
80p 14

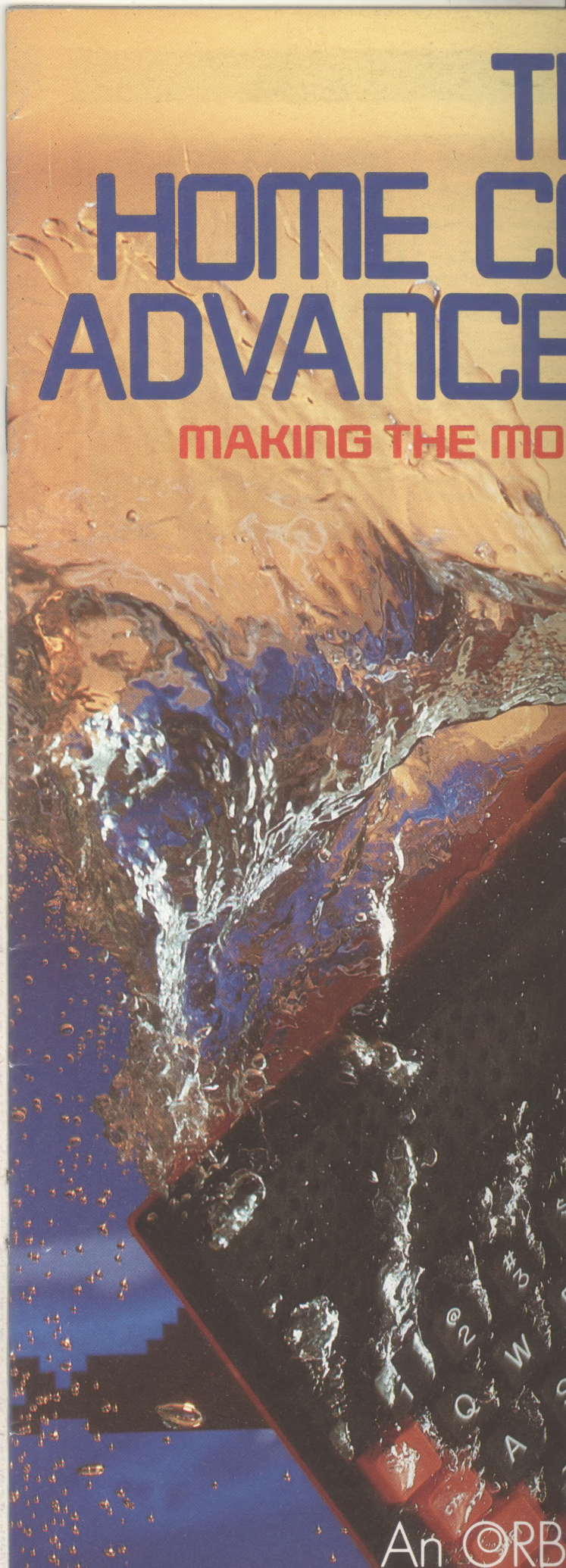
THE HOME COMPUTER ADVANCED COURSE

MAKING THE MOST OF YOUR MICRO



olication

$4.50 USA & Can $1.95



WAR MACHINE — arm yourself with all the facts

War Machine gives you the inside story. Week by week this outstanding work of reference and information builds into an unrivalled dossier of facts about the machines of war, their fire power and fighting tactics.

War Machine will feature:

* Over 2,500 weapons
* Nearly 2,000,000 words of clear and authoritative text written by acknowledged experts
* At least 5,000 pictures, nearly all in full colour
* Over 2,000 superb technical drawings and diagrams in full colour
* A country-by-country A-Z of Armed Forces, on land, sea and in the air, highlighting uniforms, equipment, orders of battle and tactical doctrines

Issue 1 on sale at newsagents now with issue 2 free

WORLD WAR II

Accept this 96-page first volume of WORLD WAR II **completely free** when you purchase your first issue of **War Machine**

The men, machinery, strategy and tactics are highlighted by action photographs and detailed campaign maps.

Start your collection with the first FREE volume, featuring: –

Hitler: the Making of a Dictator
Blitzkreig on Poland
Battle of the River Plate
The Fate of Norway
The Road to War
Rival Plans for the Western Front
Finland: the Winter War

Complete in 30 fortnightly volumes

Place a regular order at your newsagent's now